

The Complexity of the Comparator Circuit Value Problem

Stephen A. Cook¹, Yuval Filmus¹, and Dai Tri Man Lê¹

¹ Department of Computer Science, University of Toronto
{sacook,yuvalf,ledt}@cs.toronto.edu

Date: August 15, 2012

Abstract

Subramanian defined the complexity class CC as the set of problems log-space reducible to the comparator circuit value problem. He proved that several other problems are complete for CC, including the stable marriage problem, and finding the lexicographically first maximal matching in a bipartite graph.

We introduce universal comparator circuits, and as applications we prove alternative characterizations of CC: As the set of problems AC^0 many-one reducible to the comparator circuit value problem, and as problems computable by uniform polynomial-size families of comparator circuits supplied with polynomially many copies of the input and its negation. We also show that CC is closed under AC^0 ‘circuit’ reductions (i.e. reductions given by a uniform family of AC^0 circuits with oracle gates making queries to other CC problems), and that the corresponding function class FCC is closed under composition.

Subramanian showed that $NL \subseteq CC \subseteq P$. We provide evidence that CC and NC are incomparable (so that CC is a proper subset of P), by giving oracle settings where relativized CC and relativized NC are incomparable. We also give evidence that CC and SC are incomparable.

Other results include a simpler proof of $NL \subseteq CC$, a more careful analysis showing the lexicographically first maximal matching problem and its variants are CC-complete under AC^0 many-one reductions, and an explanation of the relation between the Gale-Shapley algorithm and Subramanian’s algorithm for stable marriage.

The paper continues the previous work of Cook, Lê and Ye which focused on Cook-Nguyen style uniform proof complexity, answering several open questions raised in that paper.

1 Introduction

Comparator networks were originally introduced as a method of sorting numbers (as in Batcher’s even-odd merge sort [3]), but they are still interesting when the numbers are restricted to the Boolean values $\{0, 1\}$ (in fact, a sorting network made from comparators is valid if and only if it works on Boolean inputs). A comparator gate has two inputs p, q and two outputs p', q' , where $p' = \min\{p, q\}$ and $q' = \max\{p, q\}$. In the Boolean case (which is the one we consider) $p' = p \wedge q$ and $q' = p \vee q$. A comparator circuit (i.e. network) is presented as a set of m horizontal lines in which the m inputs are presented at the left ends of the lines and the m outputs are presented at the right ends of the lines, and in between there is a sequence of comparator gates, each represented as a vertical arrow connecting some wire w_i with some wire w_j as shown in Fig. 1. These arrows divide each wire into segments, each of which gets a Boolean value. The values of wires w_i and w_j after the arrow are the comparator outputs of the values of wires w_i and w_j right before the arrow, with the tip of the arrow representing the maximum.

The comparator circuit value problem (CCV) is: given a comparator circuit with specified Boolean inputs, determine the output value of a designated wire. To turn this into a complexity class it seems natural to use a reducibility notion that is weak but robust. Thus we define CC to consist of those problems (uniform) AC^0 many-one-reducible to CCV. (Subramanian [17] studied the complexity of CCV using log-space reducibility, but fortunately our class CC is closed under log-space reducibility as well as AC^0 -reducibility.) From [17] we have

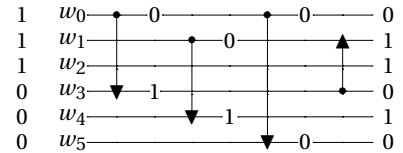


Figure 1

$$NL \subseteq CC \subseteq P, \tag{1.1}$$

where NL is nondeterministic log space. The last inclusion is obvious because CCV is a special case of the monotone circuit value problem, which is clearly in P. However comparator circuits are more restricted than monotone Boolean circuits, because each comparator gate output has fan-out one. We conjecture that $CC \subsetneq P$, and further we conjecture that CC and NC are incomparable. Here NC is the class of problems computed by uniform circuit families of polynomial size and polylog depth; intuitively NC is polylog parallel time. Similar to (1.1) we have

$$NL \subseteq NC \subseteq P.$$

In Section 5 we prove results supporting our conjecture that CC is incomparable with NC.

The complexity class CC has a number of disparate complete problems, including the comparator circuit value problem (CCV), the lexicographically first maximal matching problem (LFMM), and the stable marriage problem (SM) [12, 17]. (The first author outlined a proof that LFMM is complete under NC^1 -reductions in unpublished notes from 1983.) The SM problem is especially interesting: introduced by Gale and Shapley in 1962 [8], it has since been used to pair medical interns with hospital residencies in the USA. SM can be stated as follows: Given n men and n women, each with a complete ranking according to preference of all n members of the opposite sex, find a complete matching of the men and women such that there are no two people of opposite sex who would both rather have each other than their current partners. Gale and Shapley proved that such a ‘stable’ matching always exists, although it may not be unique.

Other interesting CC-complete problems include the stable roommate problem [17], the telephone connection problem [15], the problem of predicting internal diffusion-limited aggregation clusters from theoretical physics [13], and the decision version of the hierarchical clustering problem [9]. We refer to [17] for other CC-complete problems.

The present paper continues the research initiated in [11, 5], in which two of the present authors participated. The former work introduced a formal theory VCC which captures the complexity class CC (in the style of Cook and Nguyen [4]), and showed that Subramanian’s results are formalizable in the theory. On the way some of the proofs were simplified. In the present paper we resolve some important complexity-theoretic questions left open in [11]:

- We introduce a notion of relativized CC and prove that it is incomparable with relativized NC (Section 5).
- We show that CC is closed under AC^0 ‘circuit’ reductions (Section 3), thus identifying the three complexity classes defined in [11].
- We clarify the connection between the Gale-Shapley algorithm for stable marriage and Subramanian’s algorithm (Section 7).

Concerning the first item above, NC is often characterized informally as the class of problems which can be solved rapidly in parallel (i.e. in polylog time) using polynomially many processors. The lexicographically first maximal matching problem is in CC, but the obvious algorithm for solving it is sequential (accumulate a matching by successively adding the first unmatched edge which doesn’t touch any edge in the current matching), and we do not know of any parallel algorithm for this which approaches polylog time. On the other hand, the problem of raising an $n \times n$ integer matrix to the n th power is in NC^2 , but we do not know how to solve it using polynomial size comparator circuits, even if \neg gates are allowed.

The apparent reason that comparator circuits are limited in their computing power compared to Boolean circuits is their limited fan-out: each comparator gate has two inputs and just two outputs (\wedge and \vee). Further, if either input value to a comparator gate is ‘flipped’ (i.e. changed from 0 to 1 or from 1 to 0) then exactly one of its outputs is flipped. This generalizes to the whole circuit: If the input to any wire in a comparator circuit (possibly with \neg -gates) is flipped then at every layer in the circuit, including the output, exactly one wire is flipped. Thus flipping an input generates a unique *flip-path* through the circuit from one input wire to one output wire.

Proving either half of the conjecture ($NC \not\subseteq CC$ or $CC \not\subseteq NC$) would require a major breakthrough in complexity theory, so instead we prove a suitable relativized version of the conjecture. For this we allow oracle comparator circuits to have oracle gates, as well as comparator gates and \neg gates.

Paper organization

In Section 2 we define the basic concepts, including CC and its complete problems.

In Section 3 we introduce the notion of a universal comparator circuit. As applications we prove several results showing the robustness of the class CC. We show that the the class FCC of functions associated with CC is closed under composition, and hence CC is closed under AC^0 ‘circuit’ reductions (also called simply AC^0 reductions [2, 4]); these reductions are given by a uniform family of AC^0 circuits with oracle gates making queries to other CC problems. We also characterize CC in the style of other circuit classes such as NC and AC: a problem is in CC if and only if it is computed by some uniform polynomial size family of comparator circuits (where the circuit inputs are allowed repeated copies of the problem input bits and their negations).

In Section 4 we give a technical overview of the remaining sections of the paper in order to present these within the first ten pages.

In Section 5 we define a notion of relativized CC and prove that relativized CC is incomparable with relativized NC. This of course implies that relativized CC is strictly contained in relativized P. We also argue that CC and SC might be incomparable.

In Section 6 we prove that the lexicographically first maximal matching problem and its variants are complete for CC under AC^0 many-one reductions. We also claimed a proof that the lexicographically first maximal matching problem is complete for CC under AC^0 many-one reductions in [11], but there was a gap in that proof.

In Section 7 we show that the stable marriage problem is complete for CC, using Subramanian’s algorithm [16, 17]. We show that Subramanian’s fixed-point algorithm, which uses three-valued logic, is related to the Gale-Shapley algorithm via an intermediate *interval algorithm*. The latter algorithm also explains the provenance of three-valued logic: an interval partitions a person’s preference list into *three* parts, so we use three values $\{0, *, 1\}$ to encode these three parts of a preference list.

In Appendix ?? we include a simple proof that CC contains NL.

2 Preliminaries

2.1 Notation

We use lower case letters, e.g. x, y, z, \dots , to denote unary arguments and upper case letters, e.g. X, Y, Z, \dots , to denote binary string arguments. For a binary string X , we write $|X|$ to denote the length of X .

2.2 Function classes and search problems

A complexity class consists of relations $R(X)$, where X is a binary string argument. Given a class of relations C , we associate a class FC of functions $F(X)$ with C as follows. We require these functions to be p -bounded, i.e., $|F(X)|$ is bounded by a polynomial in $|X|$. Then we define FC to consist of all p -bounded string functions whose bit graphs are in C . (Here the *bit graph* of $F(X)$ is the relation $B_F(i, X)$ which holds iff the i th bit of $F(X)$ is 1.)

Most of the computational problems we consider here can be nicely expressed as decision problems (i.e. relations), but the stable marriage problem is an exception, because in general a given instance has more than one solution (i.e. there is more than one stable marriage). Thus the problem is properly described as a search problem. A *search problem* Q_R is a multivalued function with graph $R(X, Z)$, so $Q_R(X) = \{Z \mid R(X, Z)\}$.

The search problem is *total* if the set $Q_R(X)$ is non-empty for all X . The search problem is a *function problem* if $|Q_R(X)| = 1$ for all X . A function $F(X)$ *solves* Q_R if $F(X) \in Q_R(X)$ for all X . We will be concerned only with total search problems in this paper.

2.3 Reductions

Let C be a complexity class. A relation $R_1(X)$ is C many-one reducible to a relation $R_2(Y)$ (written $R_1 \leq_m^C R_2$) if there is a function F in FC such that $R_1(X) \leftrightarrow R_2(F(X))$.

A search problem $Q_{R_1}(X)$ is C many-one reducible to a search problem $Q_{R_2}(Y)$ if there are functions G, F in FC such that $G(X, Z) \in Q_{R_1}(X)$ for all $Z \in Q_{R_2}(F(X))$.

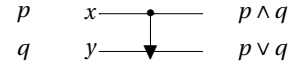
Here we are mainly interested in the cases that C is either AC^0 or L (log space). We also need a generalization of AC^0 many-one reducibility called simply AC^0 reducibility (see [2]) and denoted by \leq^{AC^0} . A function or relation is AC^0 reducible to a collection \mathcal{L} of functions and relations if it can be computed by a uniform polynomial size constant depth family of circuits which have unbounded fan-in gates computing functions and relations from \mathcal{L} (i.e. ‘oracle gates’), in addition to Boolean gates.

We note that standard small complexity classes including AC^0 , TC^0 , NC^1 , NL and P (as well as their corresponding function classes) are closed under AC^0 -reductions.

2.4 Ccv and its complexity class

A *comparator gate* is a function $C : \{0, 1\}^2 \rightarrow \{0, 1\}^2$ that takes an input pair (p, q) and outputs a pair $(p \wedge q, p \vee q)$. Intuitively, the first output in the pair is the smaller bit among the two input bits p, q , and the second output is the larger bit.

We will use the graphical notation on the right to denote a comparator gate, where x and y denote the names of the wires, and the direction of the arrow denotes the direction to which we move the larger bit as shown in the picture.



A *comparator circuit* can be defined as a directed acyclic graph consisting of: *input nodes* with in-degree zero and out-degree one, *output nodes* with in-degree one and out-degree zero, and *internal nodes* with in-degree two and out-degree two. Each internal node represents a comparator gate, with one out-edge labeled AND and the other labeled OR. If there are m input nodes then there must be m output nodes, and the circuit computes a function $f : \{0, 1\}^m \rightarrow \{0, 1\}^m$ in the obvious way.

Under this definition each comparator circuit can be represented by m horizontal wires that carry bit values (see Fig. 1 on Page 1), where m is the number of input nodes. Each comparator gate is represented by a vertical arrow connecting two of the wires. The arrowhead (representing the OR of the two inputs) can be chosen at will to point up or down, but the decision affects which wire future gates connect to. To see this, topologically sort the internal nodes of the graph (representing the comparator gates), and arrange the corresponding arrows in order from left to right. The left endpoints of the wires represent the m input nodes, and the gates can be placed one by one from left to right, each connecting the appropriate wires (determined by looking back to the last output gate touching the wire).

In this paper we present a comparator circuit by specifying its representation as horizontal wires connected by arrows, as explained in the previous paragraph. We encode the circuit by a triple (m, n, X) , where m is the number of wires and n is the number of gates, and X encodes a sequence of n pairs $\langle i, j \rangle$ with $0 \leq i, j < m$, where each pair $\langle i, j \rangle$ encodes a comparator gate that swaps the values of the wires i and j iff the value on wire i is greater than the value of wire j . For technical reasons, we also allow “dummy” gates of the form $\langle i, i \rangle$, which do nothing.

► **Definition 1.** The comparator circuit value problem (CCV) is the decision problem: Given a comparator circuit and an assignment of bits to the input nodes, decide whether a designated wire outputs one. By default, we often let the designated wire be the 0th wire of a circuit.

The complexity class CC is the class of decision problems that are AC^0 many-one reducible to CCV. A decision problem R is CC -complete if the respective class is the closure of R under AC^0 many-one reductions.

In our definition of comparator circuit each comparator gate can point in either direction, up or down (see Fig. 1). As mentioned earlier, an equivalent circuit with the same number of wires and gates can always be constructed so that all gates point up, or all gates point down, or in fact each gate can be made to point up or down arbitrarily. Transforming a circuit to one of these forms (with the same number of gates) cannot necessarily be done by an AC^0 function, but it is not hard to show the following.

► **Proposition 2.** *The CCV problem with the restriction that all comparator gates point in the same direction is CC-complete.*

Proof. Suppose we have a gate on the left of Fig. 2 with the arrow pointing upward. We can construct a circuit that outputs the same values as those of x and y , but all the gates will now point downward as shown on the right of Fig. 2.



■ **Figure 2**

It is not hard to see that the wires x_1 and y_1 in this new comparator circuit will output the same values as the wires x and y respectively in the original circuit. For the general case, we can simply make copies of all wires for each layer of the comparator circuit, where each copy of a wire will be used to carry the value of a wire at a single layer of the circuit. Then apply the above construction to simulate the effect of each gate. Note that additional comparator gates are also needed to forward the values of the wires from one layer to another, in the same way that we use the gate $\langle y_0, y_1 \rangle$ to forward the value carried in wire y_0 to wire y_1 in the above construction.

To carry this out in AC^0 , one way would be to add a complete copy of all wires for every comparator gate in the original circuit. Each new wire has input 0. For each original gate, first put in gates copying the values to the new wires, and either put in the construction in Fig. 2 or if the gate points down, simply put a copy of the gate in the original circuit. ◀

2.5 The stable marriage problem

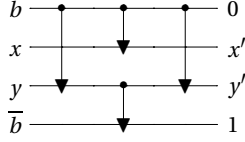
An instance of the stable marriage problem (SM), proposed by Gale and Shapley [8] in the context of college admissions, is given by a number n (specifying the number of men and the number of women), together with a preference list for each man and each woman specifying a total ordering on all people of the opposite sex. The goal of SM is to produce a perfect matching between men and women, i.e., a bijection from the set of men to the set of women, such that the following *stability* condition is satisfied: there are no two people of opposite sex who like each other more than their current partners. Such a stable solution always exists, but it may not be unique. Thus SM is a search problem (see Section 2.2), rather than a decision problem.

However there is always a unique *man-optimal* and a unique *woman-optimal* solution. In the man-optimal solution each man is matched with a woman whom he likes at least as well as any woman that he is matched with in any stable solution. Dually for the woman-optimal solution. Thus we define the *man-optimal stable marriage decision problem* (MOSM) as follows: given an instance of the stable marriage problem together with a designated man-woman pair, determine whether that pair is married in the man-optimal stable marriage. We define the *woman-optimal stable marriage decision problem* (WOSM) analogously.

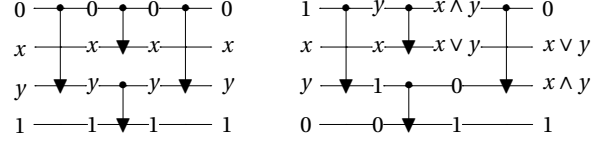
We show here that the search version and the decision versions are computationally equivalent, and each is complete for CC. Section 7.1 shows how to reduce the lexicographically first maximal matching problem (which is complete for CC) to the SM search problem, and Section 7.3 shows how to reduce both the MOSM and WOSM problems to CCV.

3 Universal comparator circuits

Here we present a construction of universal comparator circuits. The key idea is a *gadget* consisting of a comparator circuit with four wires and four gates which allows a conditional application of a comparator to two of its inputs x, y , depending on whether a bit b is 0 or 1. The other two inputs are \bar{b} and b (see Fig. 3). The comparator is applied only when $b = 1$ (see Fig. 4).



■ **Figure 3** Conditional comparator gadget



■ **Figure 4** Operation of conditional comparator gadget

In order to simulate a single arbitrary comparator in a circuit with m wires we put in $m(m-1)$ gadgets in a row, for the $m(m-1)$ possible comparators. Simulating n comparators requires $m(m-1)n$ gadgets.

Thus there is an AC^0 function $UNIV$ such that if m, n are arbitrary parameters, then $U = UNIV(m, n) = \langle m', n', U' \rangle$ is a universal circuit which simulates all comparator networks with at most m wires and at most n comparators, where the number of wires in U is $m' = 2m(m-1)n + m$ (note that the original m wires are common to all of the gadgets) and the number of gates is $n' = 4m(m-1)n$.

The AC^0 function $INPUT(C, Y) = Y'$ computes the input bits Y' for the universal circuit $U = UNIV(m, n)$, where $C = \langle \hat{m}, \hat{n}, X \rangle$ with $\hat{m} \leq m$ and $\hat{n} \leq n$. Then U with input Y' simulates the circuit C with input Y . We may arrange the universal circuit so that the \hat{m} wires of the original circuit C correspond to wires number $0, 1, \dots, \hat{m}-1$ in $UNIV(m, n)$ (and the remaining input wires specified by Y' code the inputs to the gadgets in U to simulate the comparator gates of C). From this construction, the following theorem follows.

► **Theorem 3.** *The circuit $UNIV(m, n)$ has the intended universal property. In other words, the comparator circuit $UNIV(m, n)$ on input $INPUT(C, Y)$ outputs on its first \hat{m} wires precisely the outputs of the \hat{m} wires of the original circuit C on input Y .*

The next result is an important application of universal comparator circuits. It tells us that the class CC can be characterized in terms of uniform circuit families, as in the definitions of the complexity classes NC^k and AC^k .

Let $f_{CCV}(x, y, X, Y)$ be the 0/1-function that returns the value of the 0th wire of the comparator circuit with x wires and y comparator gates encoded in X taking as input the binary string Y . By a slight abuse of notation we will write $f_{CCV}(C, Y)$ instead of $f_{CCV}(x, y, X, Y)$, where $C = \langle x, y, X \rangle$.

► **Theorem 4.** *For each relation $R(X)$ in CC there is a family $\{C_k^R\}_{k \in \mathbb{N}}$ of comparator circuits described by AC^0 functions of k such that the 0th wire of C_k^R outputs $R(X)$ when supplied with copies of $X(i), \neg X(i)$ for $i < k$ as inputs. More precisely, there are AC^0 functions $CIRCUIT^R$ and IN^R such that*

$$|X| \leq k \rightarrow [R(X) \leftrightarrow f_{CCV}(CIRCUIT^R(k), IN^R(k, X)) = 1]$$

where $CIRCUIT^R(k)$ encodes a comparator circuit with n gates and $Y = IN^R(k, X)$ consists of (for some polynomial $p = p(k)$) p copies of $X(i)$ and p copies of $\neg X(i)$, for $i = 0, 1, \dots, k-1$.

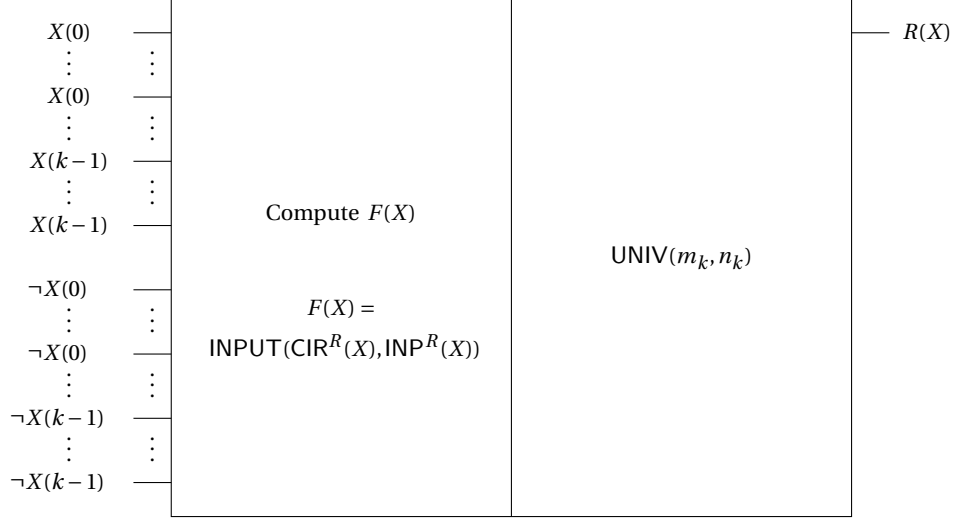
Proof. First observe that the theorem holds in case the relation $R(X)$ is in AC^0 . This is because an AC^0 circuit is easily converted to a tree whose internal nodes are AND or OR gates with fan-out one (which can be converted to comparator gates) and whose leaves have inputs of the form $IN^R(k, X)$ as described in the theorem. (No universal circuit is needed for this.)

If $F(X)$ is an AC^0 function, then its bit graph $B_F(i, X)$ is an AC^0 relation. Hence the above construction can be used to construct an AC^0 -uniform comparator circuit family $CIRCUIT^F(k)$ which on input $IN^F(k, X)$ of the form described in the theorem, outputs the bits of $F(X)$. This construction will be applied to the function $F(X)$ defined in (3.1) below.

Now suppose $R(X)$ is in CC . Then $R(X)$ is AC^0 many-one reducible to CCV (see Definition 1). Hence there are AC^0 functions $CIR^R(X)$ and $INP^R(X)$ such that

$$R(X) \leftrightarrow CCV(CIR^R(X), INP^R(X)).$$

Now given k , choose m_k and n_k such that for all X with $|X| \leq k$, if $\langle m, n, X' \rangle = CIR^R(X)$ then $m \leq m_k$ and $n \leq n_k$. We may choose m_k and n_k to be polynomials in k , because CIR^R is an AC^0 function. Then we define the



■ **Figure 5** The construction of a comparator circuit for any relation $R(X)$ in CC for a fixed k .

circuit $\text{CIRCUIT}^R(k)$ in the theorem to consist of the universal circuit $\text{UNIV}(m_k, n_k)$ preceded by a comparator circuit $\text{CIRCUIT}^F(k)$ computing the AC^0 function

$$F(X) = \text{INPUT}(\text{CIR}^R(X), \text{INP}^R(X)) \quad (3.1)$$

for $|X| \leq k$, where INPUT is as in Theorem 3. The outputs of this circuit are connected to the inputs of $\text{UNIV}(m_k, n_k)$. The function $\text{IN}^R(k, X)$ is the same as $\text{IN}^F(k, X)$, which supplies inputs to $\text{CIRCUIT}^F(k)$ (see Fig. 5).

Thus by (3.1) and Theorem 3, the 0th wire of $\text{UNIV}(m_k, n_k)$ outputs $R(X)$. ◀

► **Corollary 5.** *The function class FCC is closed under composition.*

Proof. We must show that if $G(X)$ and $H(X)$ are in FCC then so is $H \circ G(X)$. Recall that a function $G(X)$ is in FCC iff $G(X)$ is polynomially bounded and its bit graph $B_G(i, X)$ is in CC. Thus by Theorem 4 and its proof, a function $G(X)$ is in FCC iff there is an AC^0 -uniform family $\{C_k^G\}_{k \in \mathbb{N}}$ such that for $|X| \leq k$, when polynomially many copies of $X(i)$ and $\neg X(i)$, $i < k$ are fed as inputs to the circuit C_k^G using the AC^0 function IN^G , then the outputs of the bottom wires of C_k^G are the first $p(k)$ bits of $G(X)$, where $p(k)$ is a polynomial upper bound on $|G(X)|$ for $|X| \leq k$.

The circuit $C_k^{H \circ G}$ can be constructed from polynomially many copies of comparator circuits C_k^G stacked vertically (computing many copies of $G(X)$) that serve as inputs to the circuit $C_{p(k)}^H$ computing H . Actually the circuit $C_{p(k)}^H$ also needs negations of the bits of $G(X)$ as inputs. These are easily supplied, using the fact that if C is a comparator circuit and C' is the result of flipping each gate in C (interchanging AND and OR gates), then (by De Morgan's Laws) the output bits of C' with input Y' are the negations of the output bits of C with input Y , where Y' is the string of negations of the inputs Y . ◀

The following theorem shows that even though CC is defined as the closure of the CCv problem under AC^0 many-one reducibility, CC is also closed under the stronger AC^0 reducibility as defined in Section 2.3.

► **Theorem 6.** *CC is closed under AC^0 reductions.*

Proof. Theorem IX.1.7 in [4] implies that a function is AC^0 reducible to FCC if and only if it can be obtained from FCC by finitely many applications of function composition and *string comprehension*. The latter is explained by the following definition: For a function $f(x)$ which produces only unary outputs and may contain other arguments, the *string comprehension* of f is the function $F(y)$ (which outputs binary strings) satisfying

$$F(y)(i) \leftrightarrow \exists x \leq y \ i = f(x).$$

Thus, it suffices to show that FCC is closed under composition and string comprehension. We know FCC is closed under composition by Corollary 5, so it remains to show that FCC is closed under string composition. We can show this directly from the definition of FCC, without referring to any of the previous theorems. We just observe that a collection of $y+1$ instances of CCV determining whether $i = f(x)$ for $x = 0, 1, \dots, y$ can be combined by AC^0 functions of y to form an instance of CCV determining $F(y)(i)$. ◀

Corollary 5 and the fact that $NL \subseteq CC$ (see Appendix A for a simple proof of $NL \subseteq CC$) give us the following result.

► **Theorem 7.** *CC is closed under many-one NL reductions.*

Proof. This follows from the following three facts: The function class FCC is closed under composition (by Corollary 5), $FNL \subseteq FCC$, and a decision problem is in CC if and only if its characteristic function is in FCC. ◀

4 Technical overview

In this section we summarize the main results and techniques used in the remaining sections of the paper so that these ideas are presented within the first ten pages.

4.1 Oracle separations

In Section 5 we show that the relativized versions of NC and CC are incomparable. Our strategy is to construct functions *depending only on the oracle α* that are computable in one relativized complexity class but not in the other. We think of the oracle α as a length-preserving function with n inputs and n outputs. In order to avert possible criticism, we make sure that the separation works even under the promise that α has the property that if one input bit is changed, then exactly one output bit is changed. (This property is satisfied by all comparator circuits.) An oracle satisfying this property is called *strictly 1-Lipschitz*.

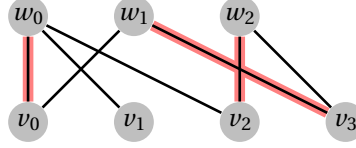
A function in relativized CC but not in relativized NC. We use an idea taken from [1]: the required function involves iterating α on the constant input 0^n , n times. This is trivially implementable in relativized CC. An information-theoretic argument from [1] shows that a depth- k circuit can only “know” the value of the k th iteration of α . Much more work (and a modified function) is needed to get a separation with a strictly 1-Lipschitz oracle.

A function in relativized NC^2 but not in relativized CC. The basic weakness of CC exploited here is the fan-out restriction. Suppose the oracle α , given n inputs, yields only $n/2$ outputs. We can feed the output back into another instance of α by using two copies of each output bit. Intuitively, a comparator circuit computing the m th iteration of α needs $2^m - 1$ copies of α , or alternatively 2^n copies of α (and a complicated circuit analyzing the output). When $m = \Omega(\log^2 n)$, this construction requires a super-polynomial size comparator circuit computing the m th iteration of α . On the other hand, for $m = O(\log^2 n)$, the m th iteration can be easily computed in relativized NC^2 .

While we were unable to prove that the particular function just described is hard for comparator circuits, we are able to prove a $\min(2^n, 2^{m-1})$ lower-bound for a related function. The crucial property of comparator circuits we use is the *flip-path* property:

If one input wire is changed, then (given that α is 1-Lipschitz) there is a unique path through the circuit tracing the effect of the original flip.

We use a Gray code to order the possible n -bit outputs of the oracle and study the effects of the 2^n flip-paths generated as the definition of the oracle is successively changed.



■ **Figure 6** The thick edges form the lfm-matching of the above bipartite graph.

4.2 Lexicographically first maximal matching

Let $G = (V, W, E)$ be a bipartite graph, where $V = \{v_i\}_{i=0}^{m-1}$, $W = \{w_i\}_{i=0}^{n-1}$ and $E \subseteq V \times W$. The *lexicographically first maximal matching* (lfm-matching) is the matching produced by successively matching each vertex v_0, \dots, v_{m-1} to the least vertex available in W (see Fig. 6 for an example). We refer to V as the set of bottom nodes and W as the set of top nodes.

In Section 6, we show that two decision problems concerning the lfm-matching of a bipartite graph are CC-complete under AC^0 many-one reductions. The lfm-matching problem (LFMM) is to decide if a designated edge belongs to the lfm-matching of a bipartite graph. The vertex version of lfm-matching problem (vLFMM) is to decide if a designated top node is matched in the lfm-matching of a bipartite graph G .

The proof showing that vLFMM is CC-complete can be summarized as follows.

- **vLFMM is in CC.** The definition of the problem outlines an algorithm. It turns out that this algorithm can be implemented using comparator circuits. To make the reduction uniform, we use dummy gates (comparator gates that compare a wire to itself).
- **vLFMM is CC-hard.** We use a simple gadget reduction, in which a comparator gate with inputs p_0, q_0 and outputs p_1, q_1 is represented by four top vertices p_0, q_0, p_1, q_1 and two bottom auxiliary vertices. We keep the invariant that the value of an input or output is 1 iff its corresponding top vertex is matched.

We then observe that the reduction from vLFMM to CCV works even if we restrict the degree of the bipartite graphs to at most 3. Thus vLFMM with degree at most 3 is already complete for CC.

To show that LFMM is CC-complete, it turns out that we need the following intermediate CC-complete problem. For comparator circuits with negation gates, we allow negation gates to appear on any wire. The comparator circuit value problem with negation gates (CCV \neg) is: given a comparator circuit with negation gates and input assignment, and a designated wire, decide if that wire outputs 1. We can extend the above two constructions to show that LFMM is AC^0 many-one reducible to CCV \neg and CCV is AC^0 many-one reducible to LFMM, even when the maximum degree is at most 3. Thus LFMM with degree at most 3 is CC-complete.

4.3 Algorithms for stable marriage

Gale and Shapley [8] gave a simple iterative algorithm for solving the stable marriage problem. Subramanian [16, 17] gave a completely different algorithm which computed both the man-optimal and the woman-optimal stable marriages as fixed-points of some iteration; see Feder [6, 7] for more on this point of view. Subramanian's algorithm shows that the stable marriage problem is in CC. It was pointed out in [11] that Subramanian's algorithm can be conveniently presented using three-valued logic (adding a value $*$ representing *unknown*), which can be implemented using “double-rail” logic (see Section 7.2). A simple gadget reduction from LFMM shows that the stable marriage problem is CC-complete (see Section 7.1).

In Section 7.3, we demystify Subramanian's algorithm by presenting a sequence of algorithms starting with the standard Gale-Shapley algorithm, and ending with Subramanian's algorithm. The original Gale-Shapley algorithm computes only the man-optimal stable marriage. Our first step is symmetrizing the algorithm so that it computes both the man-optimal and the woman-optimal stable marriages. Both algorithms can be seen as keeping track of possible partners for each person. The second step is a novel algorithm, the *interval algorithm*, in which the possible partners of a person p at each given iteration form an interval within the preference list of p .

The interval algorithm can be implemented as a fixed-point iteration in three-valued logic. The idea is that the interval splits the preference list into three parts, and this tripartition can be encoded (in a slightly non-obvious way) using three-valued logic. The fixed-point iteration cannot be implemented using comparator circuits since some values are used more than once. However there is a way to simplify the iteration so that it can be implemented using comparator circuits. The new iteration emulates the old iteration in a time-delayed fashion.

4.4 Proof of $NL \subseteq CC$

In Appendix ?? we show how to solve the directed reachability problem in CC. Our proof, also appearing in [11], is much simpler than the original one from [16, 12]. The idea is to drop n pebbles into the source vertex. Each pebble travels along the *lexicographically first maximal path* starting from the source vertex, and then the vertex at the end of this path is pebbled and excluded from the search. After n iterations, all nodes reachable from the source are pebbled, and we can check whether the target is one of them.

4.5 Open problems

Although we have shown that there are problems in relativized NC but not in relativized CC (uniform or nonuniform), it is quite possible that some of the standard problems in NC^2 that are not known to be in NL might be in (nonuniform) CC. Examples are integer matrix powering and context free languages (or more generally problems in LogCFL). Another example is the matching problem for bipartite graphs or general undirected graphs, which is in RNC^2 [10, 14] and hence in nonuniform NC^2 . It would be interesting to show that some (relativized) version of any of these problems is, or is not, in (relativized) (nonuniform) CC.

5 Oracle separations

Here we support our conjecture that the complexity classes NC and CC are incomparable by defining and separating relativized versions of the classes. (See Section 1 for a discussion of the conjecture.) Problems in relativized CC are computed by comparator circuits which are allowed to have oracle gates, as well as comparator gates and \neg gates. (By Section 6.4 the \neg gates can be eliminated.) Each oracle gate computes some function $G: \{0, 1\}^n \rightarrow \{0, 1\}^n$ for some n . We can insert such an oracle gate anywhere in an oracle comparator circuit with m wires, as long as $m \geq n$, by selecting a level in the circuit, selecting any n wires and using them as inputs to the gate (so each gate input gets one of the n distinct wires), and then the n outputs feed to some set of n distinct output wires. Note that this definition preserves the limited fan-out property of comparator circuits: each output of a gate is connected to at most one input of one other gate.

We are interested in oracles $\alpha: \{0, 1\}^* \rightarrow \{0, 1\}^*$ which are length-preserving, so $|\alpha(Y)| = |Y|$. We use the notation α_n to refer to the restriction of α to $\{0, 1\}^n$. We define the relativized complexity class $CC(\alpha)$ based on the circuit-family characterization of CC given in Theorem 4. Thus a relation $R(X, \alpha)$ is in $CC(\alpha)$ iff it is computed by a polynomial size family of comparator circuits which are allowed comparator gates, \neg -gates, and α_n oracle gates, where $n = |X|$. We consider both a uniform version (in which each circuit family satisfies a uniformity condition) and a nonuniform version of $CC(\alpha)$.

Analogous to the above, we define the relativized class $NC^k(\alpha)$ to be the class of relations $R(X, \alpha)$ computed by some family of depth $O(\log^k n)$ polynomial size Boolean circuits with \wedge , \vee , \neg , and α_n -gates (where $n = |X|$) in which \wedge -gates and \vee -gates have fan-in at most two. As above, we consider both uniform and non-uniform versions of these classes. Also $NC(\alpha) = \bigcup_k NC^k(\alpha)$.

As observed earlier, flipping one input of a comparator gate flips exactly one output. We can generalize this notion to oracles α as follows.

► **Definition 8.** A partial function $\alpha: \{0, 1\}^* \rightarrow \{0, 1\}^*$ which is length-preserving on its domain is *(weakly) 1-Lipschitz* if for all strings X, X' in the domain of α , if $|X| = |X'|$ and X and X' have Hamming distance 1, then $\alpha(X)$ and $\alpha(X')$ have Hamming distance at most 1.

A partial function $\alpha : \{0, 1\}^* \rightarrow \{0, 1\}^*$ which is length-preserving on its domain is *strictly 1-Lipschitz* if for all strings X, X' in the domain of α , if $|X| = |X'|$ and X and X' have Hamming distance 1, then $\alpha(X)$ and $\alpha(X')$ have Hamming distance *exactly* 1.

Since comparator gates compute strictly 1-Lipschitz functions, it may seem reasonable to restrict comparator oracle circuits to strictly 1-Lipschitz oracles α . Our separation results below hold whether or not we make this restriction.

Roughly speaking, we wish to prove $CC(\alpha) \not\subseteq NC(\alpha)$ and $NC^2(\alpha) \not\subseteq CC(\alpha)$. More precisely, we have the following result.

► **Theorem 9.**

- (i) *There is a relation $R_1(\alpha)$ which is computed by some uniform polynomial size family of comparator oracle circuits, but which cannot be computed by any $NC(\alpha)$ circuit family (uniform or not), even when the oracle α is restricted to be strictly 1-Lipschitz.*
- (ii) *There is a relation $R_2(\alpha)$ which is computed by some uniform $NC^2(\alpha)$ circuit family which cannot be computed by any polynomial size family of comparator oracle circuits (uniform or not), even when the oracle α is restricted to be strictly 1-Lipschitz.*

The restriction to strictly 1-Lipschitz oracles might seem severe. However the following lemma shows how to extend every 1-Lipschitz function to a strictly 1-Lipschitz function. As a result, it is enough to prove a relaxed version of Theorem 9 where the oracles are restricted to be only (weakly) 1-Lipschitz.

► **Lemma 10.** *Suppose f is a 1-Lipschitz function. Define $g(X)$ by adding a ‘parity bit’ in front of $f(X)$ as follows:*

$$g(X) = [\text{parity}(X) \oplus \text{parity}(f(X))]f(X).$$

Then g is strictly 1-Lipschitz.

Proof. Suppose $d(X, Y) = 1$. Clearly $d(g(X), g(Y)) \leq 2$. On the other hand, $\text{parity}(g(X)) = \text{parity}(X)$, and hence $d(g(X), g(Y))$ is odd. We conclude that $d(g(X), g(Y)) = 1$. ◀

Given a relation $S(\beta)$ which separates two relativized complexity classes even when the oracle β is restricted to 1-Lipschitz functions, define a new relation $R(\alpha) = S(\text{chop}(\alpha))$, where $\text{chop}(\alpha)_n(X)$ results from $\alpha_{n+1}(0X)$ by chopping off the leading bit. Given an oracle β which is 1-Lipschitz, define a new oracle α by

$$\alpha_{n+1}(bX) = [\text{parity}(bX) \oplus \text{parity}(\beta_n(X))] \beta_n(X).$$

Lemma 10 shows that α is strictly 1-Lipschitz. Notice that $R(\alpha) = S(\beta)$. Hence $R(\alpha)$ separates the two relativized complexity classes even when the oracle α is restricted to strictly 1-Lipschitz functions.

Henceforth we will prove the relaxed version of Theorem 9 in which the oracles are only restricted to be (weakly) 1-Lipschitz.

5.1 Proof of item (i) of Theorem 9

It turns out that item (i) is easy to prove if we require the $NC(\alpha)$ circuit family to work on all length-preserving oracles α , and not just 1-Lipschitz oracles. This is a consequence of the next proposition, which follows from the proof of [1, Theorem 14], and states that the ℓ th iteration of an oracle requires a circuit with oracle nesting depth ℓ to compute.

► **Definition 11.** The *nesting depth* of an oracle gate G in an oracle circuit is the maximum number of oracle gates (counting G) on any path in the circuit from an input (to the circuit) to G .

► **Proposition 12.** *Let $d, n > 0$ and let $C(\alpha)$ be a circuit with any number of Boolean gates but with fewer than 2^n α_n -gates such that the nesting depth of any α_n -gate is at most d . If the circuit correctly computes the first bit of α_n^ℓ (the ℓ th iteration of α_n), and this is true for all oracles α_n , then $\ell \leq d$.*

We apply Proposition 12 with $d = n$, and conclude that the first bit of α_n^n cannot be computed in $\text{NC}(\alpha)$. But α_n^n obviously can be computed in $\text{CC}(\alpha)$ by placing n oracle gates α_n in series. This proves item (i) without the 1-Lipschitz restriction.

For the proof of Proposition 12 we use the following definition and lemma from [1].

► **Definition 13.** A partial function $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$ is called ℓ -sequential if (abbreviating 0^n by $\mathbf{0}$)

$$\mathbf{0}, f(\mathbf{0}), f^2(\mathbf{0}), \dots, f^\ell(\mathbf{0})$$

are all defined, but $f^\ell(\mathbf{0}) \notin \text{dom}(f)$.

Note that in Definition 13 it is necessarily the case that $\mathbf{0}, f(\mathbf{0}), f^2(\mathbf{0}), \dots, f^\ell(\mathbf{0})$ are distinct.

► **Lemma 14.** Let $n \in \mathbb{N}$ and $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$ be an ℓ -sequential partial function. Let $M \subset \{0, 1\}^n$ be such that $|\text{dom}(f) \cup M| < 2^n$. Then there is an $(\ell + 1)$ -sequential extension $f' \supseteq f$ with $\text{dom}(f') = \text{dom}(f) \cup M$.

Proof. Let $Y \in \{0, 1\}^n \setminus (M \cup \text{dom}(f))$. Such a Y exists by our assumption on the cardinality of $M \cup \text{dom}(f)$. Let f' be f extended by setting $f'(x) = Y$ for all $x \in M \setminus \text{dom}(f)$. This f' is as desired.

Indeed, assume that $\mathbf{0}, f'(\mathbf{0}), \dots, f'^{\ell+1}(\mathbf{0}), f'^{\ell+2}(\mathbf{0})$ are all defined. Then, since $Y \notin \text{dom}(f')$, it follows that all the $\mathbf{0}, f'(\mathbf{0}), \dots, f'^{\ell+1}(\mathbf{0})$ have to be different from Y . Hence these values have already been defined in f . But this contradicts the assumption that f was ℓ -sequential. ◀

Proof of Proposition 12. We use f to stand for the oracle function α_n . Assume that such a circuit computes $f^\ell(\mathbf{0})$ correctly for all oracles. We have to find a setting for the oracle that witnesses $\ell \leq d$.

By induction on $k \geq 0$ we define partial functions $f_k: \{0, 1\}^n \rightarrow \{0, 1\}^n$ with the following properties.

- $f_0 \subseteq f_1 \subseteq f_2 \subseteq \dots$
- The size $|\text{dom}(f_k)|$ of the domain of f_k is at most the number of oracle gates of nesting depth k or less.
- f_k determines the values of all oracle gates of nesting depth k or less.
- f_k is k -sequential.

We can take f_0 to be the totally undefined function, since $f^0(\mathbf{0}) = \mathbf{0}$ by definition, so f_0 is 0-sequential. For the induction step let M be the set of all strings Y of length n such that Y is queried by an oracle gate at level k . Let f_{k+1} be a $k+1$ -sequential extension of f_k to domain $\text{dom}(f_k) \cup M$ according to Lemma 14.

For $k = d$ we get the desired bound. As f_d already determines the values of all gates, the output of the circuit is already determined, but $f^{d+1}(\mathbf{0})$ is still undefined and we can define it in such a way that it differs from the first bit of the output of the circuit. ◀

Now we are ready to prove item (i) for the case when oracles are restricted to be 1-Lipschitz.

Notation. We use T to stand for both a bit string and the number it represents in binary. The i th bit of T is $\text{bit}(T, i)$; the least significant bit (lsb) is bit 1. For a bit b , $b^{(m \text{ times})}$ is the bit b repeated m times. The Hamming weight of a string X is $\|X\|$. The Hamming distance between X and Y is $d(X, Y) = \|X \oplus Y\|$. The length of X is $|X|$.

► **Definition 15.** Let $n = 2^\ell$, and define $m = 2n + 1$. Given $f: \{0, 1\}^{m\ell+n} \rightarrow \{0, 1\}^n$, define the slice functions

$$f_T(X) = f(\text{bit}(T, 1)^{(m \text{ times})} \dots \text{bit}(T, \ell)^{(m \text{ times})} X), \text{ where } |T| = \ell \text{ and } |X| = n.$$

Define the iterations

$$X_0 = \mathbf{0}^{(n \text{ times})}, \quad X_{T+1} = f_T(X_T).$$

Finally, define

$$F = \text{bit}\left(X_{\lfloor \sqrt{n} \rfloor}, 1\right).$$

► **Lemma 16.** *The function $F = F(f)$ can be computed using a uniform family of comparator circuits of size polynomial in n which use f as an oracle.¹*

Proof. Obvious. ◀

If f ignores its first $m\ell$ input bits then F is the first bit of of the \sqrt{n} -th iteration of f , and hence by Proposition 12 any subexponential size circuit computing F requires depth \sqrt{n} . In the rest of this section we will show that even if f is assumed to be 1-Lipschitz, F cannot be computed by any circuit with only polynomially many oracle gates which are nested only polylogarithmically deep.

The first step is to reduce the problem of constructing a 1-Lipschitz f to the problem of constructing 1-Lipschitz f_0, \dots, f_{n-1} .

► **Definition 17.** Let f be a function as in Definition 15. Let $R_1 \dots R_\ell X$ be an input to f , where $|R_i| = m$, $|X| = n$. Suppose R_i contains z_i zeroes and o_i ones. Define $t_i = 0$ if $z_i > o_i$ and $t_i = 1$ if $z_i < o_i$ (one of these must happen since m is odd). Let $x_i = \min(z_i, o_i)$ and $x = \max_i x_i$. The values t_1, \dots, t_ℓ define a string T . We say that $R_1 \dots R_\ell X$ belongs to the blob $B(T, X)$, and is at distance x from the center string $t_1^{(m \text{ times})} \dots t_\ell^{(m \text{ times})} X$. Thus the blobs form a partition of the domain $\{0, 1\}^{m\ell+n}$ of f .

We say that f is *blob-like* if for all R_1, \dots, R_ℓ, X , with T as defined above,

$$f(R_1 \dots R_\ell X) = f_T(X) \wedge (0^{(x \text{ times})} 1^{(n-x \text{ times})}). \quad (5.1)$$

(Here we use bitwise \wedge .) In words, the value of f at a point R which is at distance x from the center of some blob B is equal to the value of f at the center of the blob, with the first x bits set to zero.

We say that f is a *blob-like partial function* if it is a partial function whose domain is a union of blobs, and inside each blob it satisfies (5.1).

Note that the values at centers of blobs are unconstrained by (5.1) because then $x = 0$.

► **Lemma 18.** *If f is blob-like and f_T is 1-Lipschitz for all $0 \leq T < n$ then f is 1-Lipschitz.*

Proof. Let $R_1 \dots R_\ell, X$ be an input to f . We argue that if we change a bit in the input, then at most one bit changes in the output. If we change a bit of X , then this follows from the 1-Lipschitz property of the corresponding f_T . If we change a bit of R_i without changing T , then we change x by at most 1, and so at most one bit of the output is affected. Finally, if we change a bit of R_i and this does change T , then we must have had (without loss of generality) $z_i = n, o_i = n + 1$, and we changed a 1 to 0 to make $z_i = n + 1, o_i = n$. In both inputs, $x = n$, and so the output is **0** in both cases. ◀

The second step is to find a way to construct 1-Lipschitz functions from $\{0, 1\}^n$ to itself, given a small number of constraints.

► **Lemma 19.** *Suppose $g: \{0, 1\}^n \rightarrow \{0, 1\}^n$ is a partial function, and $g(P) = \mathbf{0}$ for all $P \in \text{dom}(g)$. Let X be a point of Hamming distance at least d from any point in $\text{dom}(g)$. Then for every Y of Hamming weight at most d , we can extend g to a 1-Lipschitz total function satisfying $g(X) = Y$.*

Proof. Given X, Y , define $h(Z)$ to be Y with the first $\min(d(Z, X), \|Y\|)$ ones changed to zeros. We have $h(X) = Y$ since $d(X, X) = 0$. For $P \in \text{dom}(g)$, $d(P, X) \geq d$ implies $h(P) = \mathbf{0}$, using $\|Y\| \leq d$. Therefore h extends g . On the other hand, h is 1-Lipschitz since changing a bit of the input Z can change $d(Z, X)$ by at most 1, and so at most one bit of the output is affected. ◀

Finally, we need a technical lemma about the volume of Hamming balls.

¹ We can pad the output of f with $m\ell$ zeros so that f has the same number of outputs as inputs.

► **Definition 20.** Let n, d be given. Then $V(n, d)$ is the number of points in $\{0, 1\}^n$ of Hamming weight at most d , that is

$$V(n, d) = \sum_{k \leq d} \binom{n}{k}.$$

► **Lemma 21.** For $d \geq 0$, $V(n, d+1)/V(n, d) \leq n+1$.

Proof. Each point in $V(n, d+1)$ is either already a point of $V(n, d)$, or it can be obtained by taking a point of $V(n, d)$ and changing one bit from 0 to 1. Conversely, for each point of $V(n, d)$, a bit can be changed from 0 to 1 in at most n different ways. ◀

► **Corollary 22.** If $V(n, d) \geq r \geq 1$ then there exists $d' \geq 0$ such that

$$r \leq \frac{V(n, d)}{V(n, d')} < (n+1)r.$$

Proof. Let d' be the maximum number satisfying $r \leq V(n, d)/V(n, d')$. Since $r \leq V(n, d) = V(n, d)/V(n, 0)$, such a number exists. On the other hand,

$$\frac{V(n, d)}{V(n, d')} \leq (n+1) \frac{V(n, d)}{V(n, d'+1)} < (n+1)r. \quad \blacktriangleleft$$

We are now ready to prove the main lower bound, which implies item (i) of Theorem 9.

► **Theorem 23.** Let $a > 0$ be given. For large enough n , every circuit $C(f)$ with at most n^a oracle gates, nested less than \sqrt{n} deep, fails to compute F for some 1-Lipschitz function f .

Proof. Put $T_{\max} = \lfloor \sqrt{n} \rfloor - 1$. Let $d_0, \dots, d_{T_{\max}}$ be a sequence of positive integers satisfying

$$\frac{V(n, d_T)}{V(n, d_{T+1} - 1)} > n^a, \quad 0 \leq T < T_{\max}. \quad (5.2)$$

Such numbers exist whenever $2^n \geq (n+1)^{T_{\max}(a+1)}$, which holds when n is large enough. Indeed, we will construct such a sequence inductively using Corollary 22, keeping the invariant

$$V(n, d_T) \geq \frac{2^n}{(n+1)^{T(a+1)}}.$$

For the base case, $d_0 = n$ certainly satisfies the invariant. Given d_T , use Corollary 22 with $d = d_T$ and $r = (n+1)^a$. Since $V(n, d_T) \geq 2^n / (n+1)^{T(a+1)} \geq (n+1)^{a+1}$ for large n , the corollary supplies us with d' satisfying

$$(n+1)^a \leq \frac{V(n, d_T)}{V(n, d')} < (n+1)^{a+1}.$$

Let $d_{T+1} = d' + 1$. This certainly satisfies condition (5.2), and the invariant is satisfied since

$$V(n, d_{T+1}) > V(n, d') > \frac{V(n, d_T)}{(n+1)^{a+1}} \geq \frac{2^n}{(n+1)^{(T+1)(a+1)}}.$$

We will define the function f in T_{\max} stages, similar to the proof of Proposition 12, except we use the notation $f^{(k)}$ instead of f_k . At every stage the function $f^{(k)}$ will be a blob-like partial function which defines the output of every oracle gate in $C(f)$ of nesting depth k or less. The starting point is $f^{(0)}$, which is the empty function. At stage k we will define the partial function $f^{(k+1)}$, which extends $f^{(k)}$, keeping the following invariants:

- $f^{(0)} \subseteq f^{(1)} \subseteq f^{(2)} \subseteq \dots$
- $f^{(k)}$ is a blob-like partial function.
- For $T < k$, $f_T^{(k)}$ is a total 1-Lipschitz function.

- For $T \geq k$, at any point P at which $f_T^{(k)}$ is defined it is equal to $\mathbf{0}$ and some gate in $C(f)$ of nesting depth k or less has its input in the blob $B(T, P)$. Hence $|\text{dom}(f_T^{(k)})| \leq n^a$.
- X_k is defined by $f^{(k)}$.
- $f_k^{(k)}(X_k)$ is undefined.
- (*) $d(P, X_k) \geq d_k$ for any $P \in \text{dom}(f_k^{(k)})$.
- $f^{(k)}$ determines that the output of every oracle gate of nesting depth k or less.

It is easy to verify that the empty function $f^{(0)}$ satisfies the invariants. The function $f^{(T_{\max})}$ determines the output of the circuit $C(f)$. However, $X_{\lfloor \sqrt{n} \rfloor} = f_{T_{\max}}^{(T_{\max})}(X_{T_{\max}})$ is undefined. We can extend $f^{(T_{\max})}$ to a 1-Lipschitz function in two different ways: Put $f_T = \mathbf{0}$ for $T > T_{\max}$, and let $f_{T_{\max}}$ be either (1) the constant zero function, or (2) the function which is zero everywhere except for $f_{T_{\max}}(X_{T_{\max}}) = 0^{(n-1 \text{ times})}1$. Since F is different in these two extensions, the circuit fails to compute F correctly in one of them.

It remains to show how to define $f^{(k+1)}$ given $f^{(k)}$. Let \mathbb{G} be the set of oracle gates of nesting depth exactly $k+1$. For any $G \in \mathbb{G}$ whose input belongs to a blob $B(T, X)$ for $T > k$, if $f_T^{(k)}(X)$ is undefined, then define $f^{(k+1)}$ so that it extends $f^{(k)}$ and is $\mathbf{0}$ on the entire blob $B(T, X)$ (this is a blob-like assignment). Let $A = \text{dom}(f_{k+1}^{(k+1)})$; note that $|A| \leq n^a$. Condition (5.2) implies

$$V(n, d_{k+1} - 1)|A| < V(n, d_k),$$

and so there is a point Y of Hamming weight at most d_k which is of distance at least d_{k+1} from each point in A . Define $f_k^{(k+1)}(X_k) = Y$ (so $X_{k+1} = Y$), and extend $f_k^{(k+1)}$ to a total 1-Lipschitz function using Lemma 19 with $d = d_k$ (use invariant (*)). Then extend $f^{(k+1)}$ to a blob-like partial function using (5.1). It is routine to verify that the invariants hold for $f^{(k+1)}$. ◀

- Remark. ■ A more natural target function is $F' = \text{bit}(X_n, 1)$. We can easily modify the proof of Theorem 23 to handle this function. We set the first $n - \lfloor \sqrt{n} \rfloor$ functions $f^{(0)}, \dots, f^{(n - \lfloor \sqrt{n} \rfloor - 1)}$ to be constant, and then run the proof from that point on.
- An even more natural target function has an unstructured f as input, and $F'' = \text{bit}(f^{(N)}(\mathbf{0}), 1)$. We leave open the question of whether the method can be adapted to work in this case.
 - We have previously shown how to construct F''' that separates CC and NC even under the restriction that the oracle be strictly 1-Lipschitz. The function F''' basically ignores one of the outputs of the oracle while iterating it. It is possible to slightly modify the proof of Theorem 23 so that it directly applies to F even under the restriction that the oracle is strictly 1-Lipschitz. We leave this modification as an exercise to the reader.

5.2 Proof of item (ii) of Theorem 9

Here we exploit the 1-Lipschitz property of comparator gates and \neg -gates by using oracles which are weakly 1-Lipschitz, so that all gates in the relativized circuits have this property.

Let $n, m, d \in \mathbb{N}$, with $d \geq 3$. For each $k \in [m]$ and $i \in [n]$, let $a_i^k : \{0, 1\}^{dn} \rightarrow \{0, 1\}$ be a Boolean oracle with dn input bits. Let $A^k = (a_1^k, \dots, a_n^k)$. We define a function $y = f[A^1, \dots, A^m]$ as follows:

$$x_i^k = a_i^k(\overbrace{x_1^{k+1}, \dots, x_1^{k+1}}^{d \text{ times}}, \dots, \overbrace{x_n^{k+1}, \dots, x_n^{k+1}}^{d \text{ times}}), \quad k \in [m], i \in [n], \quad (5.3)$$

$$x_i^{m+1} = 0, \quad i \in [n], \quad (5.4)$$

$$y = x_1^1 \oplus \dots \oplus x_n^1. \quad (5.5)$$

As stated the oracle a_i^k has dn inputs and just one output, but we can make it fit our convention that an oracle gate has the same number of outputs as inputs simply by assuming that the gate has an additional $dn - 1$ outputs which are identically zero.

Note that the function computed by such an oracle is necessarily (weakly) 1-Lipschitz.

Let $X^k = (x_1^k, \dots, x_n^k)$ and $A^k = (a_1^k, \dots, a_n^k)$. Note that an oracle circuit of depth $m + O(\log n)$ with mn gates can compute y simply by successively computing X_m, X_{m-1}, \dots, X_1 and computing the parity of X_1 , provided that the circuit is allowed to have gates with fan-out d . However, the fan-out restriction for comparator circuits allows us to prove the following.

► **Theorem 24.** *If $n \geq 3$, then every oracle comparator circuit computing $y = f[A^1, \dots, A^m]$ has at least*

$$\min(2^n, (d-2)^{m-1})$$

gates.

By setting $m = \log^2 n$ and $d = 4$ this almost proves item (ii) of Theorem 9, except we need to argue that the array of oracles a_i^k can be replaced by a single oracle. Later we will show how a simple adaptation of the proof of Theorem 24 accomplishes this.

Proof of Theorem 24. Fix an oracle comparator circuit C which computes $y = f[A^1, \dots, A^m]$.

► **Definition 25.** We say that an input (z_1, \dots, z_{dn}) to some oracle a_i^k in C is *regular* if it has the form of the inputs in (5.3); that is if $z_{(a-1)d+b} = z_{(a-1)d+c}$ for all $a \in [n]$ and $b, c \in [d]$. We say that an oracle a_i^k is *regular* if $a_i^k(Z) = 0$ for all irregular inputs Z .

Note that any irregular oracle a_i^k can be replaced by an equivalent regular oracle which does not affect (5.3).

► **Definition 26.** Let g be the total number of any of the gates a_i^k in the circuit C . For a given assignment to the oracles, a particular gate a_i^k is *active* in C if its input is as specified by (5.3, 5.4). Let g_k be the expected total number of active gates a_1^k, \dots, a_n^k in C under a uniformly random *regular* setting of *all* oracles.

It is easy to see that

$$g_1 \geq n, \tag{5.6}$$

since we need at least one gate a_i^1 for each $i \in [n]$.

Let $k \in [m]$ be greater than 1. We will show that

$$g_{k-1} \leq \frac{g}{2^n} + \frac{g_k}{d-2}. \tag{5.7}$$

We use the following consequence of the (weakly) 1-Lipschitz property of all gates in the circuit: If we change the definition of some copy of some gate a_i^k at its input in the circuit C , this generates a unique flip-path which may end at some copy of some other gate, in which case we say that the latter gate *consumes* the flip-path. (The flip-path is a path in the circuit such that the Boolean value of each edge in the path is negated.)

Let G_1, \dots, G_{2^n} be a Gray code listing all strings in $\{0, 1\}^n$, where $G_1 = 0^n$. Thus the Hamming distance between any two successive strings G_i and G_{i+1} , and between G_{2^n} and G_1 , is one. Take a random regular setting of all the oracles, and let Z_1 be the value of X_k under this setting. Shift the above Gray code to form a new one Z_1, \dots, Z_{2^n} by setting $Z_t = G_t \oplus Z_1$. Then for each $t \in [2^n]$, Z_t is uniformly distributed and independent of X_ℓ for $\ell \neq k$. Thus if we change the output of A^k at its active input to Z_t , the result is again a uniformly random regular oracle setting. Let γ_t be the number of active A^k gates (i.e. any active gate of the form a_i^k for some i) after this change, and let δ_t be the number of active A^{k-1} gates after the change. Taking expectations we have for each $t \in [2^n]$

$$\mathbb{E}(\gamma_t) = g_k, \quad \mathbb{E}(\delta_t) = g_{k-1}. \tag{5.8}$$

We will change the output of A^k (at its active input) successively from Z_1 to Z_{2^n} , and consider the relationship between γ_t and δ_t . The total number of flip paths generated during the process is

$$\sum_{t=1}^{2^t-1} \gamma_t.$$

Each time an A^{k-1} gate is rendered active for the first time, we will call the gate *fresh*. Otherwise, it is *reused*. At time t , let δ'_t (δ''_t) be the number of fresh (reused) A^{k-1} gates. Thus $\delta''_1 = 0$, and

$$\delta_t = \delta'_t + \delta''_t \quad (5.9)$$

Since a given gate can be fresh at most once, we have

$$\sum_{t=1}^{2^n} \delta'_t \leq g \quad (5.10)$$

Each time an A^{k-1} gate is reused it has consumed at least $d-2$ flip-paths since the last time it was active. This is because at least d consecutive inputs must be changed from all 0's to all 1's (or vice versa), and since the gate is regular, its output will be constantly 0 during at least $d-2$ consecutive changes.

Since there must be at least as many flip-paths generated as consumed, we have

$$(d-2) \sum_{t=1}^{2^n} \delta''_t \leq \sum_{t=1}^{2^n} \gamma_t. \quad (5.11)$$

From (5.9), (5.10), (5.11) we have

$$\sum_{t=1}^{2^n} \delta_t \leq g + \frac{1}{d-2} \sum_{t=1}^{2^n} \gamma_t. \quad (5.12)$$

Now (5.7) follows from (5.12) and (5.8) by linearity of expectations.

Hence either $g > 2^n$ or

$$g_k \geq (d-2)[g_{k-1} - 1].$$

From this and (5.6) we have a recurrence whose solution shows

$$g_t \geq (d-2)^{t-1} n - \frac{(d-2)^t - (d-2)}{d-3}.$$

If $n \geq 3$ then $g_m \geq (d-2)^{m-1}$, and Theorem 24 follows. ◀

Now we change the setting in Theorem 24 so that it applies to a single oracle. The new oracle $a(k, i, x)$ is used in the same way as $a_i^k(x)$. The first two arguments can be encoded in binary or unary, and we don't care what happens when they are not "legal" (we don't require the output to be 0 unless the x argument is illegal). Define active a_i^k gates as gates whose inputs are (k, i, x) , where x is the relevant active input. We argue as before, and again conclude that if $n \geq 3$ then $g_m \geq (d-2)^{m-1}$. Hence Theorem 24 follows in the single oracle setting.

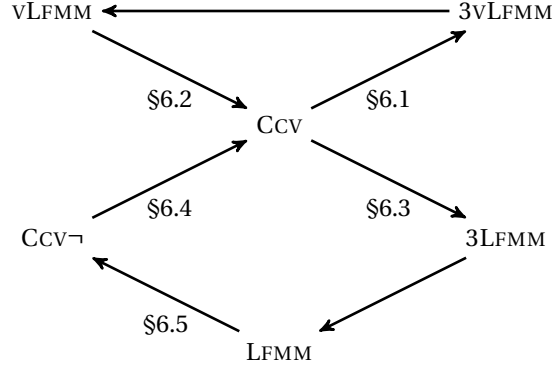
5.3 SC vs CC

Uniform SC^k is the class of problems decidable by Turing machines running in polynomial time using $O(\log^k n)$ space. Non-uniform SC^k is the class of problems solvable by circuits of polynomial size and $O(\log^k n)$ width. Just as NC and CC appear to be incomparable, it seems plausible that SC and CC are incomparable. For one direction, NL is a subclass of both CC and NC, but is conjectured not to be a subclass of SC (Savitch's algorithm takes $2^{O(\log^2)}$ time). For the other direction we can give a convincing oracle separation as follows.

We apply Theorem 24 to a problem with a padded input of length N , and set the 'real' input $n = \log^2 N$, and also $m = \log^2 N$ and $d = 4$. The theorem implies that every comparator circuit solving F has size at least

$$2^{\min(m-1, n)} = 2^{\log^2 N},$$

which is superpolynomial. Thus this padded problem is not in relativized CC.



■ **Figure 7** The label of an arrow denotes the section in which the reduction is described. Arrows without labels denote trivial reductions. All six problems are CC-complete.

However, a Turing machine M equipped with an oracle tape which can query $4\log^2 N$ bits of each oracle a_i^k can compute this padded version of F in linear time and $O(n) = O(\log^2 N)$ space, so this problem is in relativized SC^2 . The machine M proceeds by successively computing X_m, X_{m-1}, \dots, X_1 , writing each of these X_i on its work tape and then erasing the previous one. The machine computes X^k from X^{k+1} bit by bit, making a query of size $4\log^2 N$ to its query tape for each bit. (We assume that M can access its oracle in such a way that it can determine N , and hence m and n .)

6 Lexicographically first maximal matching problems are CC-complete

Let $G = (V, W, E)$ be a bipartite graph, where $V = \{v_i\}_{i=0}^{m-1}$, $W = \{w_i\}_{i=0}^{n-1}$ and $E \subseteq V \times W$. The *lexicographically first maximal matching* (lfm-matching) is the matching produced by successively matching each vertex v_0, \dots, v_{m-1} to the least vertex available in W . We refer to V as the set of bottom nodes and W as the set of top nodes.

In this section we will show that two decision problems concerning the lfm-matching of a bipartite graph are CC-complete under AC^0 many-one reductions. The lfm-matching problem (LFMM) is to decide if a designated edge belongs to the lfm-matching of a bipartite graph G . The vertex version of lfm-matching problem (vLFMM) is to decide if a designated top node is matched in the lfm-matching of a bipartite graph G . LFMM is the usual way to define a decision problem for lfm-matching as seen in [12, 17]; however, as shown in Sections 6.1 and 6.2, the vLFMM problem is even more closely related to the CCV problem.

We will show that the following two more restricted lfm-matching problems are also CC-complete. We define 3LFMM to be the restriction of LFMM to bipartite graphs of degree at most three. We define 3vLFMM to be the restriction of vLFMM to bipartite graphs of degree at most three.

To show that the problems defined above are equivalent under AC^0 many-one reductions, it turns out that we also need the following intermediate problem. A negation gate flips the value on a wire. For comparator circuits with negation gates, we allow negation gates to appear on any wire (see the left diagram of Fig. 11 below for an example). The comparator circuit value problem with negation gates (CCV¬) is: given a comparator circuit with negation gates and input assignment, and a designated wire, decide if that wire outputs 1.

All reductions in this section are summarized in Fig. 7.

6.1 $CCV \leq_m^{AC^0} 3vLFMM$

By Proposition 2 it suffices to consider only instances of CCV in which all comparator gates point upward. We will show that these instances of CCV are AC^0 many-one reducible to instances of 3vLFMM, which consist of bipartite graphs with *degree at most three*.

The key observation is that a comparator gate on the left below closely relates to an instance of 3VLFMM on the right. We use the top nodes p_0 and q_0 to represent the values p_0 and q_0 carried by the wires x and y respectively before the comparator gate, and the nodes p_1 and q_1 to represent the values of x and y after the comparator gate, where a top node is matched iff its respective value is one.



If nodes p_0 and q_0 have not been previously matched, i.e. $p_0 = q_0 = 0$ in the comparator circuit, then the edges $\langle x, p_0 \rangle$ and $\langle y, q_0 \rangle$ are added to the lfm-matching. So the nodes p_1 and q_1 are not matched. If p_0 has been previously matched, but q_0 has not, then edges $\langle x, p_1 \rangle$ and $\langle y, q_0 \rangle$ are added to the lfm-matching. So the node p_1 will be matched but q_1 will remain unmatched. The other two cases are similar.

Thus, we can reduce a comparator circuit to the bipartite graph of an 3VLFMM instance by converting each comparator gate into the “gadget” described above. We will describe our method through an example, where we are given the comparator circuit in Fig. 8.

We divide the comparator circuit into vertical layers 0, 1, 2 as shown in Fig. 8. Since the circuit has three wires a, b, c , for each layer i , we use six nodes, including three top nodes a_i, b_i and c_i representing the values of the wires a, b, c respectively, and three bottom nodes a'_i, b'_i, c'_i , which are auxiliary nodes used to simulate the effect of the comparator gate at layer i .

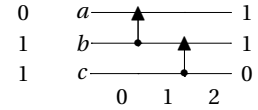
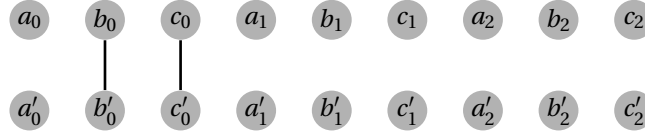


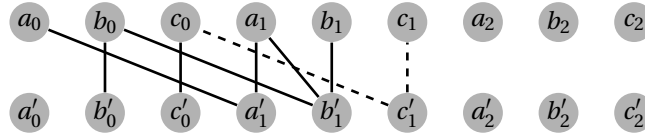
Figure 8

Layer 0: This is the input layer, so we add an edge $\{x_i, x'_i\}$ iff the wire x takes

input value 1. In this example, since b and c are wires taking input 1, we need to add the edges $\{b_0, b'_0\}$ and $\{c_0, c'_0\}$.

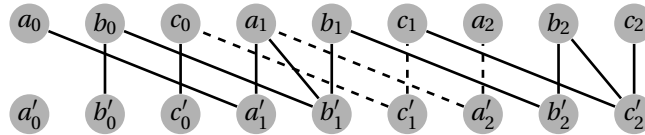


Layer 1: We then add the gadget simulating the comparator gate from wire b to wire a as follows.



Since the value of wire c does not change when going from layer 0 to layer 1, we can simply propagate the value of c_0 to c_1 using the pair of dashed edges in the picture.

Layer 2: We proceed very similarly to layer 1 to get the following bipartite graph.



Finally, we can get the output values of the comparator circuit by looking at the “output” nodes a_2, b_2, c_2 of this bipartite graph. We can easily check that a_2 is the only node that remains unmatched, which corresponds exactly to the only zero produced by wire a of the comparator circuit in Fig. 8.

It remains to argue that the construction above is an AC^0 many-one reduction. We observe that each gate in the comparator circuit can be independently reduced to exactly one gadget in the bipartite graph that simulates the effect of the comparator gate; furthermore, the position of each gadget can be easily calculated from the position of each gate in the comparator circuit using very simple arithmetics.

6.2 $\text{vLFMM} \leq_m^{\text{AC}^0} \text{CCV}$

Consider the instance of vLFMM consisting of the bipartite graph in Fig. 9. Recall that we find the lfm-matching by matching the bottom nodes x, y, z successively to the first available node on the top. Hence we can simulate the matching of the bottom nodes to the top nodes using the comparator circuit on the right of Fig. 9, where we can think of the moving of a 1, say from wire x to wire a , as the matching of node x to node a in the original bipartite graph. In this construction, a top node is matched iff its corresponding wire in the circuit outputs 1.



Figure 9

Note that we draw bullets without any arrows going out from them in the circuit to denote dummy gates, which do nothing. These dummy gates are introduced for the following technical reason. Since the bottom nodes might not have the same degree, the position of a comparator gate really depends on the structure of the graph, which makes it harder to give a direct AC^0 reduction. By using dummy gates, we can treat the graph as if it is a complete bipartite graph, the missing edges represented by dummy gates. This can easily be shown to be an AC^0 reduction from vLFMM to CCV . Together with the reduction from Section 6.1, we get the following theorem.

► **Theorem 27.** *The problems CCV , 3vLFMM and vLFMM are equivalent under AC^0 many-one reductions.*

6.3 $\text{CCV} \leq_m^{\text{AC}^0} 3\text{LFMM}$

We start by applying the reduction $\text{CCV} \leq_m^{\text{AC}^0} 3\text{vLFMM}$ of Section 6.1 to get an instance of 3vLFMM , and notice that the degrees of the top “output” nodes of the resulting bipartite graph, e.g. the nodes a_2, b_2, c_2 in the example of Section 6.1, have degree at most two. Now we show how to reduce such instances of 3vLFMM (i.e. those whose designated top vertices have degree at most two) to 3LFMM . Consider the graph G with degree at most three and a designated top vertex b of degree two as shown on the left of Fig. 10. We extend it to a bipartite graph G' by adding an additional top node w_t and an additional bottom node w_b , alongside two edges $\{b, w_b\}$ and $\{w_t, w_b\}$, as shown in Fig. 10. Observe that the degree of the new graph G' is at most three.

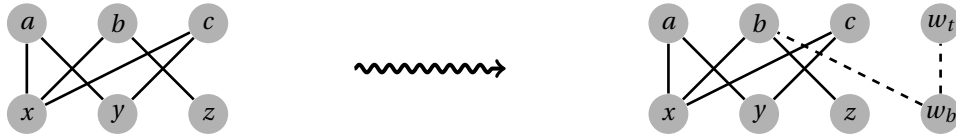
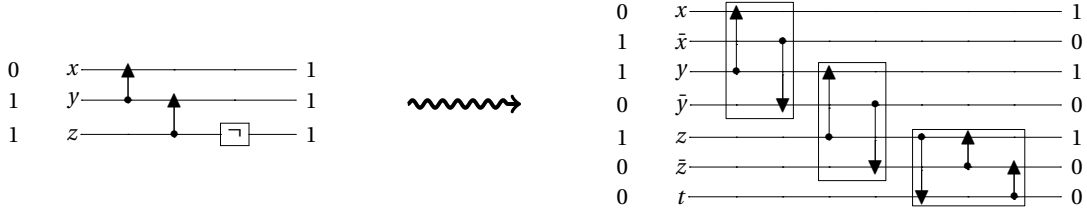


Figure 10

We treat the resulting bipartite graph G' and the edge $\{w_t, w_b\}$ as an instance of 3LFMM . It is not hard to see that the vertex b is matched in the lfm-matching of the original bipartite graph G iff the edge $\{w_t, w_b\}$ is in the lfm-matching of the new bipartite graph G' .

6.4 $\text{CCV} \neg \leq_m^{\text{AC}^0} \text{CCV}$

Recall that a comparator circuit value problem with negation gates ($\text{CCV}\neg$) is the task of deciding, given a comparator circuit with negation gates and an input assignment, whether a designated wire outputs one. It should



■ **Figure 11** Successive gates on the left circuit correspond to successive boxes of gates on the right circuit.

be clear that CCV is a special case of $\text{CCV}\neg$ and hence AC^0 many-one reducible to $\text{CCV}\neg$. Here, we show the nontrivial direction that $\text{CCV}\neg \leq_m^{\text{AC}^0} \text{CCV}$. Our proof is based on Subramanian's idea from [17].

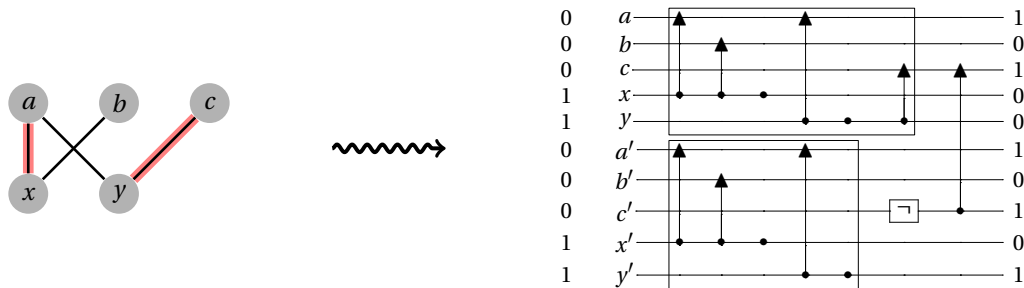
The reduction is based on “double-rail” logic. Given an instance of $\text{CCV}\neg$ consisting of a comparator circuit with negation gates C with its input I and a designated wire s , we construct an instance of CCV consisting of a comparator circuit C' with its input I' and a designated wire s' as follows. For every wire w in I we put in two corresponding wires, w and \bar{w} , in C' . We define the input I' of C' such that the input value of \bar{w} is the negation of the input value of w . We want to fix things so that the value carried by the wire \bar{w} at each layer is always the negation of the value carried by w . For any comparator gate $\langle y, x \rangle$ in C we put in C' the gate $\langle y, x \rangle$ followed by the gate $\langle \bar{x}, \bar{y} \rangle$. It is easy to check using De Morgan's laws that the wires x and y in C' carry the corresponding values of x and y in C , and the wires \bar{x} and \bar{y} in C' carry the negations of the wires x and y in C .

The circuit C' has one extra wire t with input value 0 to help in translating negation gates. For each negation gate on a wire, says z in the example from Fig. 11, we add three comparator gates $\langle z, t \rangle$, $\langle \bar{z}, z \rangle$, $\langle t, \bar{z} \rangle$ as shown in the right circuit of Fig. 11. Thus t as a temporary “container” that we use to swap the values carried by the wires z and \bar{z} . Note that the swapping of values of z and \bar{z} in C' simulates the effect of a negation in C . Also note that after the swap takes place, the value of t is restored to 0. (The more straightforward solution of simply switching the wires z and \bar{z} does not result in an AC^0 many-one reduction.)

Finally note that the output value of the designated wire s in C is 1 iff the output value of the corresponding wire s in C' with input I' is 1. Thus we set the designated wire s' in I' to be s .

6.5 $\text{LFMM} \leq_m^{\text{AC}^0} \text{CCV}\neg$

Consider an instance of LFMM consisting of the bipartite graph on the left of Fig. 12, and a designated edge $\{y, c\}$. Without loss of generality, we can safely ignore all top vertices occurring after c , all bottom vertices occurring after y , and all the edges associated with them, since they are not going to affect the outcome of the instance. Using the construction from Section 6.2, we can simulate the matching of the bottom nodes to the top nodes using the comparator circuit in the upper box on the right of Fig. 12.



■ **Figure 12**

We keep another running copy of this simulation on the bottom (see the wires labelled a', b', c', x', y' in Fig. 12). The only difference is that the comparator gate $\langle y', c' \rangle$ corresponding to the designated edge $\{y, c\}$ is

not added. Finally, we add a negation gate on c' and a comparator gate $\langle c', c \rangle$. We let the desired output of the CCV instance be the output of c , since c outputs 1 iff the edge $\{y, c\}$ is added to the lfm-matching. It is not hard to generalize this construction to an arbitrary bipartite graph and designated edge.

Combined with the constructions from Sections 6.1 and 6.2, we have the following corollary.

► **Corollary 28.** *The problems CCV, 3VLFMM, VLFMM, CCV \neg , 3LFMM and LFMM are equivalent under AC^0 many-one reductions.*

Since CCV \neg is complete for CC, we can use comparator circuits to decide the complement of the CCV problem: given a comparator circuit and an input assignment, does a designated wire output zero? Thus, we have the following corollary.

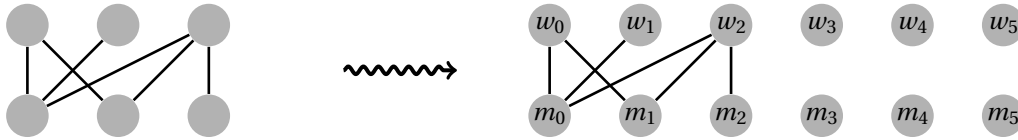
► **Corollary 29** (Subramanian [12]). *CC is closed under complementation.*

7 The SM problem is CC-complete

7.1 3LFMM is AC^0 many-one reducible to SM, MoSM and WoSM

We start by showing that 3LFMM is AC^0 many-one reducible to SM when we regard both 3LFMM and SM as search problems. (Of course the lfm-matching is the unique solution to 3LFMM formulated as a search problem, but it is still a total search problem.)

Let $G = (V, W, E)$ be a bipartite graph from an instance of 3LFMM, where V is the set of bottom nodes, W is the set of top nodes, and E is the edge relation such that the degree of each node is at most three (see the example in the figure on the left below). Without loss of generality, we can assume that $|V| = |W| = n$. To reduce it to an instance of SM, we double the number of nodes in each partition, where the new nodes are enumerated after the original nodes and the original nodes are enumerated using the ordering of the original bipartite graph, as shown in the diagram on the right below. We also let the bottom nodes and top nodes represent the men and women respectively.



It remains to define a preference list for each person in this SM instance. The preference list of each man m_i , who represents a bottom node in the original graph, starts with all the women w_j (at most three of them) adjacent to m_i in the order that these women are enumerated, followed by all the women w_n, \dots, w_{2n-1} ; the list ends with all women w_j not adjacent to m_i also in the order that they are enumerated. For example, the preference list of m_2 in our example is $w_2, w_3, w_4, w_5, w_0, w_1$. The preference list of each newly introduced man m_{n+i} simply consists of $w_0, \dots, w_{n-1}, w_n, \dots, w_{2n-1}$, i.e., in the order that the top nodes are listed. Preference lists for the women are defined dually.

Intuitively, the preference lists are constructed so that any stable marriage (not necessarily man-optimal) of the new SM instance must contain the lfm-matching of G . Furthermore, if a bottom node u from the original graph is not matched to any top node in the lfm-matching of G , then the man m_i representing u will marry some top node w_{n+j} , which is a dummy node that does not correspond to any node of G .

The above construction gives us a AC^0 many-one reduction from 3LFMM to SM as search problems, any solution of a stable marriage instance constructed by the above reduction providing us all the information to decide whether an edge is in the lfm-matching of the original 3LFMM instance. The key explanation is that every instance of stable marriage produced by the above reduction has a unique solution; thus the man-optimal solution coincides with the woman-optimal solution. Moreover, the above construction also shows that the decision version of 3LFMM is AC^0 many-one reducible to either of the decision problems MoSM and WoSM. Hence we have proven the following theorem, whose detailed proof can be found in [5].

► **Theorem 30.** *3LFMM is AC^0 many-one reducible to SM, MoSM and WoSM.*

7.2 THREE-VALUED CCV is CC-complete

In the remainder of the section, we will be occupied with developing an algorithm due to Subramanian [16, 17] that finds a stable marriage using comparator circuits, thus furnishing an AC^0 reduction from SM to CCV. To this end, it turns out to be conceptually simpler to go through a new variant of CCV, where the wires are three-valued instead of Boolean.

We define the THREE-VALUED CCV problem similarly to CCV, i.e., we want to decide, on a given input assignment, if a designated wire of a comparator circuit outputs one. The only difference is that each wire can now take either value 0, 1 or *, where a wire takes value * when its value is not known to be 0 or 1. The output values of the comparator gate on two input values p and q will be defined as follows.

$$p \wedge q = \begin{cases} 0 & \text{if } p = 0 \text{ or } q = 0 \\ 1 & \text{if } p = q = 1 \\ * & \text{otherwise.} \end{cases} \quad p \vee q = \begin{cases} 0 & \text{if } p = q = 0 \\ 1 & \text{if } p = 1 \text{ or } q = 1 \\ * & \text{otherwise.} \end{cases}$$

Clearly every instance of CCV is also an instance of THREE-VALUED CCV. We will show that every instance of THREE-VALUED CCV is AC^0 many-one reducible to an instance of CCV by using a pair of Boolean wires to represent each three-valued wire and adding comparator gates appropriately to simulate three-valued comparator gates.

► **Theorem 31.** *THREE-VALUED CCV and CCV are equivalent under AC^0 many-one reductions.*

Proof. Since each instance of CCV is a special case of THREE-VALUED CCV, it only remains to show that every instance of THREE-VALUED CCV is AC^0 many-one reducible to an instance of CCV.

First, we will describe a gadget built from standard comparator gates that simulates a three-valued comparator gate as follows. Each wire of an instance of THREE-VALUED CCV will be represented by a pair of wires in an instance of CCV. Each three-valued comparator gate on the left below, where $p, q, p \wedge q, p \vee q \in \{0, 1, *\}$, can be simulated by a gadget consisting of two standard comparator gates on the right below.



The wires x and y are represented using the two pairs of wires $\langle x_1, x_2 \rangle$ and $\langle y_1, y_2 \rangle$, and three possible values 0, 1 and * will be encoded by $\langle 0, 0 \rangle$, $\langle 1, 1 \rangle$, and $\langle 0, 1 \rangle$ respectively. The fact that our gadget correctly simulates the three-valued comparator gate is shown in the following table.

p	q	$\langle p_1, p_2 \rangle$	$\langle q_1, q_2 \rangle$	$p \wedge q$	$p \vee q$	$\langle p_1 \wedge q_1, p_2 \wedge q_2 \rangle$	$\langle p_1 \vee q_1, p_2 \vee q_2 \rangle$
0	0	$\langle 0, 0 \rangle$	$\langle 0, 0 \rangle$	0	0	$\langle 0, 0 \rangle$	$\langle 0, 0 \rangle$
0	1	$\langle 0, 0 \rangle$	$\langle 1, 1 \rangle$	0	1	$\langle 0, 0 \rangle$	$\langle 1, 1 \rangle$
0	*	$\langle 0, 0 \rangle$	$\langle 0, 1 \rangle$	0	*	$\langle 0, 0 \rangle$	$\langle 0, 1 \rangle$
1	0	$\langle 1, 1 \rangle$	$\langle 0, 0 \rangle$	0	1	$\langle 0, 0 \rangle$	$\langle 1, 1 \rangle$
1	1	$\langle 1, 1 \rangle$	$\langle 1, 1 \rangle$	1	1	$\langle 1, 1 \rangle$	$\langle 1, 1 \rangle$
1	*	$\langle 1, 1 \rangle$	$\langle 0, 1 \rangle$	*	1	$\langle 0, 1 \rangle$	$\langle 1, 1 \rangle$
*	0	$\langle 0, 1 \rangle$	$\langle 0, 0 \rangle$	0	*	$\langle 0, 0 \rangle$	$\langle 0, 1 \rangle$
*	1	$\langle 0, 1 \rangle$	$\langle 1, 1 \rangle$	*	1	$\langle 0, 1 \rangle$	$\langle 1, 1 \rangle$
*	*	$\langle 0, 1 \rangle$	$\langle 0, 1 \rangle$	*	*	$\langle 0, 1 \rangle$	$\langle 0, 1 \rangle$

Using this gadget, we can reduce an instance of THREE-VALUED CCV to an instance of CCV by doubling the number of wires, and replacing every three-valued comparator gate of the THREE-VALUED CCV instance with a gadget with two standard comparator gates simulating it.

The above construction shows how to reduce the question of whether a designated wire outputs 1 for a given instance of THREE-VALUED CCV to the question of whether a *pair* of wires of an instance of CCV output $\langle 1, 1 \rangle$.

However for an instance of CCV we are only allowed to decide whether a *single* designated wire outputs 1. This technical difficulty can be easily overcome since we can use an \wedge -gate (one of the two outputs of a comparator gate) to test whether a pair of wires outputs $\langle 1, 1 \rangle$, and output the result on a single designated wire. ◀

7.3 Algorithms for solving stable marriage problems

In this section, we develop a reduction from SM to CCV due to Subramanian [16, 17], and later extended to a more general class of problems by Feder [6, 7]. Subramanian did not reduce SM to CCV directly, but to the *network stability problem* built from the less standard X gate, which takes two inputs p and q and produces two outputs $p' = p \wedge \neg q$ and $q' = \neg p \wedge q$. It is important to note that the “network” notion in Subramanian’s work denotes a generalization of circuits by allowing a connection from the output of a gate to the input of any gate including itself, and thus a network in his definition might contain cycles. An X-network is a network consisting only of X gates under the important restriction that each X gate has fan-out exactly one for each output it computes. The network stability problem for X gates (XNS) is then to decide if an X-network has a stable configuration, i.e., a way to assign Boolean values to the wires of the network so that the values are compatible with all the X gates of the network. Subramanian showed in his dissertation [16] that SM, XNS and CCV are all equivalent under log space reductions.

We do not work with XNS in this paper since networks are less intuitive and do not have a nice graphical representation as do comparator circuits. By utilizing Subramanian’s idea, we give a direct AC^0 reduction from SM to CCV, using the three-valued variant of CCV developed in Section 7.2.

We will describe a sequence of algorithms, starting with Gale and Shapley’s algorithm, which is historically the first algorithm solving the stable marriage problem, and ending with Subramanian’s algorithm.

7.3.1 Notation

Let M denote the set of men, and W denote the set of women; both are of size n . The preference list for a person p is given by

$$\pi_1(p) >_p \pi_2(p) >_p \cdots >_p \pi_n(p) >_p \perp.$$

The last place on the list is taken by the placeholder \perp which represents p being unmatched, a situation less preferable than being matched. If p is a man then $\pi_1(p), \dots, \pi_n(p)$ are women, and vice versa.

The preference relation $>_p$ is defined by $\pi_i(p) >_p \pi_j(p)$ whenever $i < j$; we say that p prefers $\pi_i(p)$ over $\pi_j(p)$. For a set of women W_0 and a man m , the woman m prefers the most is $\max_m W_0$; if W_0 is empty, then $\max_m W_0 = \perp$. Let S be a set, then we write $q >_p S$ to denote that p prefers q to any person in S ; similarly, $q <_p S$ denotes that p prefers any person in S to q .

A marriage P is a set of pairs (m, w) which forms a perfect matching between the set of men and the set of women. In a marriage P , we let $P(p)$ denote the person p is married. A marriage is stable if there is no unstable pair (m, w) , which is a pair satisfying $w >_m P(m)$ and $m >_w P(w)$, i.e. m and w prefer each other more than their current partner.

7.3.2 Gale-Shapley

Gale and Shapley’s algorithm [8] proceeds in rounds. In the first round, each man proposes to his top woman among the ones he hasn’t proposed, and each woman selects her most preferred suitor. In each subsequent round, each rejected man proposes to his next choice, and each woman selects her most preferred suitor (including her choice from the previous round). The situation eventually stabilizes, resulting in the man-optimal stable marriage.

There are many ways to implement the algorithm. One of them is illustrated below in Algorithm 1. The crucial object is the graph G , which is a set of possible matches. Each round, each man m selects the top woman $\text{top}(m)$ currently available to him. Among all men who chose her (if any), each woman w selects the best suitor $\text{best}(w)$. Whenever any man m is rejected by his top woman w , we remove the possible match (m, w) from G .

Algorithm 1 Gale-Shapley

```

 $G \leftarrow \{(m, w) : m \in M, w \in W\}$ 
repeat
   $\text{top}(m) \leftarrow \max_m \{w : (m, w) \in G\}$  for all  $m \in M$ 
   $\text{best}(w) \leftarrow \max_w \{m : \text{top}(m) = w\}$  for all  $w \in W$ 
  Remove  $(m, \text{top}(m))$  from  $G$  whenever  $\text{best}(\text{top}(m)) \neq m$ 
until  $G$  stops changing
return  $\{(m, \text{top}(m)) : m \in M\}$ 

```

► **Lemma 32.** *Algorithm 1 returns the man-optimal stable matching, and terminates after at most n^2 rounds.*

Proof. Admissibility: If a pair (m, w) is removed from G , then no stable marriage contains the pair (m, w) . This is proved by induction on the number of pairs removed. A pair (m, w) is removed when $w = \text{top}(m) = \text{top}(m')$ for some other man m' , and $m' \succ_w m$. Suppose for a contradiction that P is a stable marriage and if $P(m) = w$. By the induction hypothesis, we know that m' can never be married to any woman w' such that $w' \succ_{m'} w$ since that edge (m', w') was removed previously. Thus $w \succeq_{m'} P(m')$. But then (m', w) would be an unstable pair, a contradiction.

Definiteness: For all men m and at all times, $\text{top}(m) \neq \perp$. For any man m , $(m, \pi_n(m))$ is never removed from G , and so $\text{top}(m)$ is always well-defined. Indeed, for each w , after each iteration $\text{best}(w)$ is non-decreasing in the preference order of w . So if $(m, w) \notin G$, $\text{best}(w) \succ_w m$. On the other hand, for any two women w and w' , if $\text{best}(w), \text{best}(w') \neq \perp$ then $\text{best}(w) \neq \text{best}(w')$. Thus, if $(m, w) \notin G$ for all $w \in W$, then best is an injective mapping from W into $M \setminus \{m\}$, contradicting the pigeonhole principle.

Completeness: The output of the algorithm is a marriage. The algorithm ends when $\text{best}(\text{top}(m)) = m$ for every m , which implies that best and top are mutually inverse bijections.

Stability: The output of the algorithm is a stable marriage. Suppose (m, w) were an unstable pair, so at the end of the algorithm, $m \succ_w \text{best}(w)$ and $w \succ_m \text{top}(m)$ (we're using the fact that top and best are inverses at the end of the algorithm). However, $m \succ_w \text{best}(w)$ implies $\text{top}(m) \neq w$, which implies $\text{top}(m) \succ_m w$.

Optimality: The output of the algorithm is the man-optimal stable marriage. This is obvious, since each man gets his best choice among all possible stable marriages.

Runtime: The algorithm terminates in n^2 iterations since at most n^2 edges can be deleted from G . ◀

Gale-Shapley has one disadvantage: it only computes the man-optimal stable matching. This is easy to rectify by symmetrizing the algorithm, resulting in Algorithm 2. While in the original algorithm, only the men propose (select their top choices), and only the women accept or reject (choose the most promising suitor), in the symmetric algorithm, both sexes participate in both tasks in parallel. The algorithm returns both the man-optimal and the woman-optimal stable marriages.

Algorithm 2 Symmetric Gale-Shapley

```

 $G \leftarrow \{(m, w) : m \in M, w \in W\}$ 
repeat
   $\text{top}(m) \leftarrow \max_m \{w : (m, w) \in G\}$  for all  $m \in M$ 
   $\text{top}(w) \leftarrow \max_w \{m : (m, w) \in G\}$  for all  $w \in W$ 
   $\text{best}(w) \leftarrow \max_w \{m : \text{top}(m) = w\}$  for all  $w \in W$ 
   $\text{best}(m) \leftarrow \max_m \{w : \text{top}(w) = m\}$  for all  $m \in M$ 
  Remove  $(m, \text{top}(m))$  from  $G$  whenever  $\text{best}(\text{top}(m)) \neq m$ 
  Remove  $(\text{top}(w), w)$  from  $G$  whenever  $\text{best}(\text{top}(w)) \neq w$ 
until  $G$  stops changing
return  $\{(m, \text{top}(m)) : m \in M\}$  and  $\{(\text{top}(w), w) : w \in W\}$  as the man-optimal and the woman-optimal stable marriages respectively

```

► **Lemma 33.** *Algorithm 2 returns the man-optimal and woman-optimal stable matchings, and terminates after at most n^2 rounds.*

Proof. The analysis is largely analogous to the analysis of the original algorithm. Every pair (m, w) removed from G belongs to no stable marriage. Furthermore, since a stable marriage exists, $\text{top}(m)$ and $\text{top}(w)$ are always defined after the algorithm finishes. At the end of the algorithm, top and best are mutually inverse bijections on $M \cup W$, hence the outputs are marriages. The same arguments as before show that the marriages returned by the algorithm are man-optimal and woman-optimal stable marriages respectively. Finally, the algorithm terminates in n^2 iterations since we can only remove at most n^2 edges G . ◀

7.3.3 Interval algorithms

At the end of Algorithm 2, for each man m , his partner in the man-optimal stable marriage is $\text{top}(m)$, while his partner in the woman-optimal stable marriage is $\text{best}(m)$. The same holds for women (with the roles of the sexes reversed). This prompts our next algorithm, Algorithm 3, which explicitly keeps track of an interval $J(p)$ of possible matches for each person p (these are intervals in the person's preference order).

At each round, each person p first picks their top choice $\text{top}(p)$. Then each person q picks their top suitor $\text{best}(q)$, if any. People over whom $\text{best}(q)$ is preferred are removed from $J(q)$. If p was rejected by his top choice $\text{top}(p)$, then $\text{top}(p)$ is removed from $J(p)$. These update rules maintain the contiguous nature of the intervals. The situation eventually stabilizes, and the algorithm returns the man-optimal and the woman-optimal stable marriages.

Algorithm 3 Interval algorithm

$J_0(m) \leftarrow W$ for all $m \in M$

$J_0(w) \leftarrow M$ for all $w \in W$

$t \leftarrow 0$

repeat

$\text{top}_t(p) \leftarrow \max_p J_t(p)$ for all $p \in M \cup W$

$\text{best}_t(q) \leftarrow \max_q \{p : q = \text{top}_t(p)\}$ for all $q \in M \cup W$

 Remove p from $J_t(q)$ whenever $p <_q \text{best}_t(q)$, for all p, q of opposite sex

 Remove $\text{top}_t(p)$ from $J_t(p)$ if $p \neq \text{best}_t(\text{top}_t(p))$

$t \leftarrow t + 1$

until $J_{t+1}(p) = J_t(p)$ for all $p \in M \cup W$

return $\{(m, \max_m J_t(m)) : m \in M\}$ and $\{(\max_w J_t(w), w) : w \in W\}$ as the man-optimal and the woman-optimal stable marriages respectively

► **Lemma 34.** *Algorithm 3 returns the man-optimal and woman-optimal stable matchings, and terminates after at most $2n^2$ rounds. Furthermore, the man-optimal and woman-optimal matchings are given by*

$$\{(m, \max_m J_t(m)) : m \in M\} \text{ and } \{(\max_w J_t(w), w) : w \in W\} \text{ respectively.}$$

Proof. Admissibility: In every stable marriage, every person p is matched to someone from $J(p)$. This is proved by induction on the number of rounds. A person q can be removed from $J(p)$ for one of two reasons: either $q <_p \text{best}(p)$, or $q = \text{top}(p)$ and $p \neq \text{best}(q)$. In the former case, if p were matched to q , then $(p, \text{best}(p))$ would be an unstable pair, since $p = \text{top}(\text{best}(p))$ implies that $\text{best}(p)$ prefers p to any other partner in $J(\text{best}(p))$. In the latter case, if p were matched to $q = \text{top}(p)$, then $(q, \text{best}(q))$ would be an unstable pair, since q prefers $\text{best}(q)$ over p by definition, and $\text{best}(q)$ prefers q as in the former case.

The remaining analysis of this algorithm is similar to the analysis of Gale-Shapley. The outputs of the algorithm are marriages, since the algorithm ends when $\text{best}(\text{top}(p)) = p$ for all p , hence top and best are inverse bijections. The marriages are stable for the same reason given for Gale-Shapley. They are man-optimal and

woman-optimal for the same reason. The number of iterations is at most $2n^2$ since there are $2n$ intervals, each of initial length n .

Finally, at the termination of the algorithm, $\text{best}(q) = \max_q J(q)$. Since top and best are inverses, this explains the dual formulas for the man-optimal and woman-optimal matchings. ◀

Our next algorithm introduces a new twist. Instead of removing $\text{top}(p)$ from $J(p)$ whenever $p \neq \text{best}(\text{top}(p))$, we remove $\text{top}(p)$ from $J(p)$ whenever $p \notin J(\text{top}(p))$ as shown in Algorithm 4. The idea is that if at some point $p \neq \text{best}(\text{top}(p))$, then $\text{best}(\text{top}(p)) \succ_{\text{top}(p)} p$, so p is removed from $J(\text{top}(p))$. At the following iteration, $\text{top}(p)$ will be removed from $J(p)$ in reciprocity. Thus, Algorithm 4 emulates Algorithm 3 with a delay of one round. We will later show that the advantage of this strange rule is the nice representation of the same algorithm in three-valued logic which can then be transformed to Subramanian's algorithm, implementable by comparator circuits.

Algorithm 4 Delayed interval algorithm

$J_0(m) \leftarrow W$ for all $m \in M$

$J_0(w) \leftarrow M$ for all $w \in W$

$t \leftarrow 0$

repeat

$\text{top}_t(p) \leftarrow \max_p J(p)$ for all $p \in M \cup W$

$\text{best}_t(q) \leftarrow \max_q \{p : q = \text{top}_t(p)\}$ for all $q \in M \cup W$

Remove p from $J_t(q)$ whenever $p <_q \text{best}_t(q)$, for all p, q of opposite sex

Remove $\text{top}_t(p)$ from $J_t(p)$ if $p \notin J_t(\text{top}_t(p))$

$t \leftarrow t + 1$

until $J_{t+1}(p) = J_t(p)$ for all $p \in M \cup W$

return $\{(m, \max_m J_t(m)) : m \in M\}$ and $\{(\max_w J_t(w), w) : w \in W\}$ as the man-optimal and the woman-optimal stable marriages respectively

► **Lemma 35.** *Algorithm 3 returns the man-optimal and woman-optimal stable matchings, and terminates after at most $2n^2$ rounds. Furthermore, the man-optimal and woman-optimal matchings are given by*

$$\{(m, \max_m J_t(m)) : m \in M\} \text{ and } \{(\max_w J_t(w), w) : w \in W\} \text{ respectively.}$$

Proof. Clearly Algorithm 4 is admissible, that is p is matched to someone from $J(p)$ in any stable matching. Furthermore, at the end of the algorithm, $p = \text{best}(\text{top}(p))$. Otherwise, there are two cases. If $p \in J(\text{top}(p))$, then p would be removed from $J(\text{top}(p))$, and the algorithm would continue. If $p \notin J(\text{top}(p))$, then $\text{top}(p)$ would be removed from $J(p)$, and the algorithm would continue; note that by definition, at the beginning of the round, $\text{top}(p) \in J(p)$.

The rest of the proof follows the one for Algorithm 3. ◀

► **Corollary 36.** *The intervals at the end of Algorithm 3 coincide with the intervals at the end of Algorithm 4.*

Proof. That follows immediately from the two formulas for the output. ◀

The delayed interval algorithm can be implemented using three-valued logic. The key is the following encoding of the intervals using matrices, which we call the *matrix representation*.

$$\mathcal{M}(m, w) = \begin{cases} 1 & \text{if } w \geq_m \max_m J(m) \\ * & \text{if } \max_m J(m) \succ_m w \geq_m \min_m J(m) \\ 0 & \text{if } \min_m J(m) \succ_m w \end{cases}$$

$$\mathcal{W}(w, m) = \begin{cases} 0 & \text{if } m \geq_w \max_w J(w) \\ * & \text{if } \max_w J(w) \succ_w m \geq_w \min_w J(w) \\ 1 & \text{if } \min_w J(w) \succ_w m \end{cases}$$

In other words, for every man m , the array $\mathcal{M}(m, \pi_1(m)), \dots, \mathcal{M}(m, \pi_n(m))$ has the form

$$1 \quad \dots \quad \boxed{1 \quad * \quad \dots \quad *} \quad 0 \quad \dots \quad 0$$

where the men whose corresponding values are contained in the box are precisely the men in $J(m)$.

For every woman w , the array $\mathcal{W}(w, \pi_1(w)), \dots, \mathcal{W}(w, \pi_n(w))$ has the form

$$0 \quad \dots \quad \boxed{0 \quad * \quad \dots \quad *} \quad 1 \quad \dots \quad 1$$

where the women whose corresponding values are contained in the box are precisely the women in $J(w)$.

Algorithm 5 is an implementation of Algorithm 4 using three-valued logic. We will show, in a sequence of steps, that at each point in time, the matrices representing the intervals in Algorithm 4 equal the matrices in Algorithm 5.

Algorithm 5 Delayed interval algorithm, three-valued logic formulation

$$\mathcal{M}_0(m, w) = \begin{cases} 1 & \text{if } w = \pi_1(m) \\ * & \text{otherwise} \end{cases}$$

$$\mathcal{W}_0(w, m) = \begin{cases} 0 & \text{if } m = \pi_1(w) \\ * & \text{otherwise} \end{cases}$$

$t \leftarrow 0$

repeat

$$\mathcal{M}_{t+1}(m, \pi_i(m)) = \begin{cases} 1 & \text{if } i = 1 \\ \mathcal{M}_t(m, \pi_{i-1}(m)) \wedge \bigwedge_{j \leq i-1} \mathcal{W}_t(\pi_j(m), m) & \text{otherwise} \end{cases}$$

$$\mathcal{W}_{t+1}(w, \pi_i(w)) = \begin{cases} 0 & \text{if } i = 1 \\ \mathcal{W}_t(w, \pi_{i-1}(w)) \vee \bigvee_{j \leq i-1} \mathcal{M}_t(\pi_j(w), w) & \text{otherwise} \end{cases}$$

$t \leftarrow t + 1$

until $\mathcal{M}_t = \mathcal{M}_{t-1}$ and $\mathcal{W}_t = \mathcal{W}_{t-1}$

$S_M \leftarrow \{(m, w) : \mathcal{M}_t(m, w) = 1 \text{ and } \mathcal{W}_t(w, m) \in \{0, *\}\}$ % man-optimal stable marriage

$S_W \leftarrow \{(m, w) : \mathcal{W}_t(w, m) = 0 \text{ and } \mathcal{M}_t(m, w) \in \{1, *\}\}$ % woman-optimal stable marriage

return S_M, S_W

First, we show that the matrices properly encode intervals.

► **Lemma 37.** *At each time t in the execution of Algorithm 5, and for each man m , the sequence*

$$\mathcal{M}_t(m, \pi_1(m)), \dots, \mathcal{M}_t(m, \pi_n(m))$$

*is non-increasing (with respect to the order $1 > * > 0$). Similarly, for each woman w , the sequence*

$$\mathcal{W}_t(w, \pi_1(w)), \dots, \mathcal{W}_t(w, \pi_n(w))$$

is non-decreasing

Proof. The proof is by induction. The claim is clearly true at time $t = 0$. For the inductive case, it suffices to analyze \mathcal{M} since \mathcal{W} can be handled dually. Furthermore, at each iteration, we “shift” each sequence $\mathcal{M}(m, \cdot)$ one step to the right and add a 1 to the left end to get the following non-increasing sequence

$$1, \mathcal{M}_{t-1}(m, \pi_1(m)), \mathcal{M}_{t-1}(m, \pi_2(m)), \dots, \mathcal{M}_{t-1}(m, \pi_{n-1}(m))$$

Then we take a component-wise AND of the above sequence with the non-increasing sequence

$$1, \mathcal{W}_{t-1}(\pi_1(m), m), (\mathcal{W}_{t-1}(\pi_1(m), m) \wedge \mathcal{W}_{t-1}(\pi_2(m), m)), \dots, (\mathcal{W}_{t-1}(\pi_1(m), m) \wedge \dots \wedge \mathcal{W}_{t-1}(\pi_{n-1}(m), m)).$$

It's not hard to check that the result is also a non-increasing sequence by the properties of three-valued logic. ◀

Second, we show that the intervals encoded by the matrices can only shrink. This is the same as saying that whenever an entry gets determined (to a value different from $*$), it remains constant.

► **Lemma 38.** *If for some time t , for some man m and some woman w , $\mathcal{M}_t(m, w) \in \{0, 1\}$, then $\mathcal{M}_s(m, w) = \mathcal{M}_t(m, w)$ for $s \geq t$. A similar claim holds for \mathcal{W} .*

Proof. We prove the claim by induction on t . Let $w = \pi_i(m)$. If $i = 1$, then the claim is trivial. Now suppose $i > 1$. If $\mathcal{M}_t(m, \pi_i(m)) = 1$, then

$$\mathcal{M}_{t-1}(m, \pi_{i-1}(m)) = \mathcal{W}_{t-1}(\pi_1(m), m) = \dots = \mathcal{W}_{t-1}(\pi_{i-1}(m), m) = 1.$$

The induction hypothesis shows that all these elements retain their values in the next iteration and hence $\mathcal{M}_{t+1}(m, \pi_{i+1}(m)) = 1$. If $\mathcal{M}_t(m, \pi_i(m)) = 0$, then at least one of these elements is equal to zero; this element retains its value in the next iteration by the induction hypothesis; hence $\mathcal{M}_{t+1}(m, \pi_{i+1}(m)) = 0$. ◀

It remains to show that the way that the underlying intervals are updated matches the update rules of Algorithm 4.

► **Lemma 39.** *At each time t , the matrix representation of the intervals in Algorithm 4 is the same as the matrices $\mathcal{M}_t, \mathcal{W}_t$ in the execution of Algorithm 5. Furthermore, both algorithms return the same marriages.*

Proof. The proof is by induction on t . The base case $t = 0$ is clear by inspection.

We now compare the update rules in some round t of both algorithms. There are two ways an interval $J(m)$ can be updated: either a woman is removed from the bottom of the interval, or a woman is removed from the top of the interval.

In the former case, a woman $\pi_i(m)$ is removed from $J_{t+1}(m)$ since $\pi_i(m) <_m \text{best}_t(m)$. Suppose $\text{best}_t(m) = \pi_j(m)$, where $j < i$. Since $m = \text{top}_t(\pi_j(m))$, we know that $\mathcal{W}_t(\pi_j(m), m) = 0$, and so

$$\mathcal{M}_{t+1}(m, \pi_i(m)) = \mathcal{M}_t(m, \pi_{i-1}(m)) \wedge \bigwedge_{j \leq i-1} \mathcal{W}_t(\pi_j(m), m) = 0. \quad (7.1)$$

Conversely, suppose $\mathcal{M}_{t+1}(m, \pi_i(m)) = 0$ while $\mathcal{M}_t(m, \pi_i(m)) = *$. Since $\mathcal{M}_t(m, \cdot)$ is non-increasing, we have $\mathcal{M}_t(m, \pi_{i-1}(m)) \neq 0$. Thus for $\mathcal{M}_{t+1}(m, \pi_i(m)) = 0$, we must have $\mathcal{W}_t(\pi_j(m), m) = 0$ for some $j < i$. Now suppose $m \neq \text{top}_t(\pi_j(m))$, then that equation (7.1) were true at an earlier time $s < t$, at which $\mathcal{M}_s(m, \pi_i(m))$ would have become 0. Hence $m = \text{top}_t(\pi_j(m))$, and $\pi_i(m)$ is removed from $J_{t+1}(m)$.

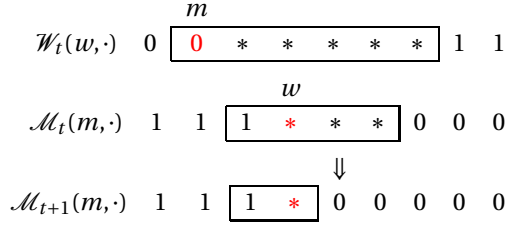
In the latter case, a woman $\pi_i(m)$ is removed from $J_{t+1}(m)$ since $\pi_i(m) = \text{top}_t(m)$ and $m \notin J_t(\pi_i(m))$. We claim that $m <_{\pi_i(m)} J(\pi_i(m))$, since otherwise $m >_{\pi_i(m)} J(\pi_i(m))$. Thus $(m, \pi_i(m))$ would be an unstable pair in any marriage produced by the algorithm (m will be matched to a woman inferior to $\text{top}_t(m) = \pi_i(m)$, and $\pi_i(m)$ will be matched to a man from $J(\pi_i(m))$ whom $\pi_i(m)$ doesn't like as much as m), and this contradicts that the algorithm produces some stable marriage. Hence we have $m <_{\pi_i(m)} J(\pi_i(m))$ and so $\mathcal{W}_t(\pi_i(m), m) = 1$. For $j \leq i-1$, the woman $\pi_j(m)$ must have been removed from $J(m)$ in the past (since women can only be removed from the top of $J(m)$ one at a time), and at that time $\mathcal{W}(\pi_j(m), m) = 1$ (just as at the current time, $\mathcal{W}_t(\pi_j(m), m) = 1$); by Lemma 38, this value stays 1. Hence

$$\mathcal{M}_{t+1}(m, \pi_{i+1}(m)) = \mathcal{M}_t(m, \pi_i(m)) \wedge \bigwedge_{j \leq i} \mathcal{W}_t(\pi_j(m), m) = 1.$$

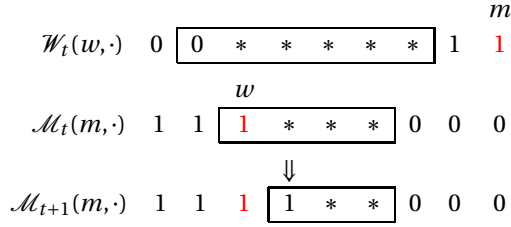
Conversely, suppose $\mathcal{M}_{t+1}(m, \pi_{i+1}(m)) = 1$ while $\mathcal{M}_t(m, \pi_{i+1}(m)) = *$. Since $\mathcal{M}_t(m, \pi_i(m)) = 1$, we must have $\pi_i(m) = \text{top}_t(m)$. Further, $\mathcal{W}_t(\pi_i(m), m) = 1$ shows that $m \notin J_t(\pi_i(m))$. Therefore $\pi_i(m)$ is removed from $J_{t+1}(m)$.

Finally, we conclude that the two algorithms return the same man-optimal and woman-optimal stable marriages by Lemma 37. ◀

Let us illustrate the workings of Algorithm 5. The following diagram illustrates a situation in which a man's interval shrinks from the bottom. The diagram illustrates $\mathcal{W}_t(w, \cdot)$, $\mathcal{M}_t(m, \cdot)$, $\mathcal{M}_{t+1}(m, \cdot)$, respectively, where $m = \text{top}_t(w)$. The two elements in red form a pair $\mathcal{M}(m, w)$, $\mathcal{W}(w, m)$.



The next diagram illustrates a situation in which a man's interval shrinks from the top. This time $w = \text{top}_t(m)$.



7.3.4 Subramanian's algorithm

Subramanian's algorithm is very similar to Algorithm 5. The latter algorithm is not implementable using comparator circuits, since (for example) the value $\mathcal{W}_t(\pi_1(m), m)$ is used in the update rule of

$$\mathcal{M}_{t+1}(m, \pi_2(m)), \dots, \mathcal{M}_{t+1}(m, \pi_n(m)).$$

Subramanian's algorithm, displayed as Algorithm 6, corrects this issue by retaining only the most important term in each conjunction and disjunction.

Algorithm 6 Subramanian's algorithm

$$\begin{aligned}
 \mathcal{M}_0(m, w) &= \begin{cases} 1 & \text{if } w = \pi_1(m) \\ * & \text{otherwise} \end{cases} \\
 \mathcal{W}_0(w, m) &= \begin{cases} 0 & \text{if } m = \pi_1(w) \\ * & \text{otherwise} \end{cases} \\
 t &\leftarrow 0 \\
 \textbf{repeat} & \\
 \mathcal{M}_{t+1}(m, \pi_i(m)) &= \begin{cases} 1 & \text{if } i = 1 \\ \mathcal{M}_t(m, \pi_{i-1}(m)) \wedge \mathcal{W}_t(\pi_{i-1}(m), m) & \text{otherwise} \end{cases} \\
 \mathcal{W}_{t+1}(w, \pi_i(w)) &= \begin{cases} 0 & \text{if } i = 1 \\ \mathcal{W}_t(w, \pi_{i-1}(w)) \vee \mathcal{M}_t(\pi_{i-1}(w), w) & \text{otherwise} \end{cases} \\
 t &\leftarrow t + 1 \\
 \textbf{until} & \mathcal{M}_t = \mathcal{M}_{t-1} \text{ and } \mathcal{W}_t = \mathcal{W}_{t-1} \\
 S_M &\leftarrow \{(m, w) : \mathcal{M}_t(m, w) = 1 \text{ and } \mathcal{W}_t(w, m) \in \{0, *\}\} & \% \text{ man-optimal stable marriage} \\
 S_W &\leftarrow \{(m, w) : \mathcal{W}_t(w, m) = 0 \text{ and } \mathcal{M}_t(m, w) \in \{1, *\}\} & \% \text{ woman-optimal stable marriage} \\
 \textbf{return} & S_M, S_W
 \end{aligned}$$

In the analysis of Algorithm 5, we already saw that Subramanian's update rule works when an entry of \mathcal{M} is set to 1. When Algorithm 5 sets an entry to 0, say $\mathcal{M}_{t+1}(m, \pi_i(m)) = 0$, it is due to $\mathcal{W}_t(\pi_j(m), m) = 0$ for some $j \leq i - 1$.

If $j = i - 1$, then Subramanian's algorithm will also set $\mathcal{M}_{t+1}(m, \pi_i(m)) = 0$. Otherwise, Subramanian's algorithm will set $\mathcal{M}_{t+1}(m, \pi_{j+1}(m)) = 0$, and this zero will propagate, so that $\mathcal{M}_{t+i-j}(m, \pi_i(m)) = 0$. This shows that in some sense, Subramanian's algorithm mimics Algorithm 5. It remains to show that Subramanian's algorithm computes the same final \mathcal{M} and \mathcal{W} as Algorithm 5.

First, we notice that the termination conditions of both algorithms are really the same. For Subramanian's algorithm, the conditions are that for $i > 1$,

$$\begin{aligned}\mathcal{M}(m, \pi_i(m)) &= \mathcal{M}(m, \pi_{i-1}(m)) \wedge \mathcal{W}(\pi_{i-1}(m), m), \\ \mathcal{W}(w, \pi_i(w)) &= \mathcal{W}(w, \pi_{i-1}(w)) \vee \mathcal{M}(\pi_{i-1}(w), w).\end{aligned}\tag{7.2}$$

For Algorithm 5, the termination conditions are

$$\begin{aligned}\mathcal{M}(m, \pi_i(m)) &= \mathcal{M}(m, \pi_{i-1}(m)) \wedge \bigwedge_{j \leq i-1} \mathcal{W}(\pi_j(m), m), \\ \mathcal{W}(w, \pi_i(w)) &= \mathcal{W}(w, \pi_{i-1}(w)) \vee \bigvee_{j \leq i-1} \mathcal{M}(\pi_j(w), w).\end{aligned}\tag{7.3}$$

► **Lemma 40.** *The matrices \mathcal{M}, \mathcal{W} at the end of Subramanian's algorithm satisfy the termination conditions of Algorithm 5, and vice versa. Moreover, these are always matrix representations of intervals.*

Proof. We observe in both algorithms the update rules guarantee that $\mathcal{M}(m, \cdot)$ is monotone non-increasing and that $\mathcal{W}(w, \cdot)$ is monotone non-decreasing, which implies

$$\mathcal{M}(\pi_{i-1}(w), w) = \bigvee_{j \leq i-1} \mathcal{M}(\pi_j(w), w), \quad \mathcal{W}(\pi_{i-1}(m), m) = \bigwedge_{j \leq i-1} \mathcal{W}(\pi_j(m), m).$$

Thus, these two termination conditions are equivalent. ◀

We call a pair of matrices $(\mathcal{M}, \mathcal{W})$ a *feasible pair* if they satisfy the equations in (7.2) or equivalently in (7.3), and furthermore $\mathcal{M}(m, \pi_1(m)) = 1$ and $\mathcal{W}(w, \pi_1(w)) = 0$ for all man m and woman w . The following lemma shows that, in some sense, Subramanian's algorithm is admissible.

► **Lemma 41.** *Let \mathcal{M}, \mathcal{W} be the matrices at the end of Subramanian's algorithm. If $\mathcal{M}(m, w) = c \neq *$ for some m, w , then $\mathcal{M}'(m, w) = c$ for any feasible pair $(\mathcal{M}', \mathcal{W}')$. Same for \mathcal{W} .*

Proof. The proof is by induction on the time t in which $\mathcal{M}_t(m, w)$ is set to c . If $t = 0$, then the claim follows from the definition of feasible pair. Otherwise, for some i we have $\mathcal{M}_t(m, \pi_i(m)) = \mathcal{M}_{t-1}(m, \pi_{i-1}(m)) \wedge \mathcal{W}_{t-1}(\pi_{i-1}(m), m)$. If $c = 1$ then $\mathcal{M}_{t-1}(m, \pi_{i-1}(m)) = \mathcal{W}_{t-1}(\pi_{i-1}(m), m) = 1$, and by induction these entries get the same value in all feasible pairs. The definition of feasible pair then implies that $\mathcal{M}'(m, \pi_i(m)) = 1$ in any feasible pair $\mathcal{M}', \mathcal{W}'$. The case when $c = 0$ can be shown similarly. ◀

The following lemma shows that we can uniquely extract a stable marriage from each 0/1-valued feasible pair, i.e. both of the matrices in the pair have 0/1 values, and vice versa.

► **Lemma 42.** *Suppose $(\mathcal{M}, \mathcal{W})$ is a 0/1-valued feasible pair. If we marry each man m to $\min_m \{w : \mathcal{M}(m, w) = 1\}$, and each woman w to $\min_w \{m : \mathcal{W}(m, w) = 0\}$, then the result is a stable marriage.*

Conversely, every stable marriage P can be encoded as a 0/1-valued feasible pair, as follows: For each man m , we put $\mathcal{M}(m, w) = 1$ if $w \succeq_m P(m)$, and $\mathcal{M}(m, w) = 0$ otherwise. For each woman w , we put $\mathcal{W}(w, m) = 0$ if $m \succeq_w P(w)$, and $\mathcal{W}(w, m) = 1$ otherwise.

The two mappings are inverses of each other.

Proof. Feasible pair implies stable marriage. Suppose $(\mathcal{M}, \mathcal{W})$ is a 0/1-valued feasible pair. We start by showing that the mapping P in the statement of the lemma is indeed a marriage.

We call a person *desperate* if he or she is married to the last choice in his or her preference list.

If a man m is not desperate then for some $1 \leq i < n$, $\mathcal{M}(m, \pi_i(m)) = 1$ and $\mathcal{M}(m, \pi_{i+1}(m)) = 0$. This can only happen if $\mathcal{W}(\pi_i(m), m) = 0$, and furthermore if $m \succ_{\pi_i(m)} m'$, then $\mathcal{W}(\pi_i(m), m') = 1$ due to $\mathcal{M}(m, \pi_i(m)) = 1$. This shows that whenever a man m is not desperate, m is married to someone in the marriage P .

If m is desperate and $\mathcal{W}(\pi_i(m), m) = 0$, then m is married as before. Otherwise, no woman is desperate, and so similarly to the previous argument, it can be shown that every woman w is married in P . However, the fact that $P(w) \neq m$ for all women w contradicts the pigeonhole principle. Thus, we conclude that P is a marriage.

It remains to show that P is stable. Suppose that (m, w) were an unstable pair in P . Then $\mathcal{M}(m, w) = 1$ and $\mathcal{W}(w, m) = 0$, and moreover $\mathcal{M}(m, w') = 1$ for some $w' \prec_m w$. Yet (7.3) shows that $\mathcal{M}(m, w') \leq \mathcal{W}(w, m)$, and we reach a contradiction.

Stable marriage implies feasible pair. Suppose m is matched to $\pi_k(m)$. Consider first the case $i \leq k$. Then $\mathcal{M}(m, \pi_i(m)) = \mathcal{M}(m, \pi_{i-1}(m)) = 1$, and we have to show that $\mathcal{W}(\pi_{i-1}(m), m) = 1$. If the latter weren't true then $(m, \pi_{i-1}(m))$ would be an unstable pair, since $\pi_{i-1}(m) \succ_m \pi_i(m)$ while $\mathcal{W}(\pi_{i-1}(m), m) = 0$ implies that $\pi_{i-1}(m)$ prefers m to every other man which is matched to her. If $i = k+1$ then $\mathcal{M}(m, \pi_i(m)) = 0$ and also $\mathcal{W}(\pi_{i-1}(m), m) = 0$. If $i > k$ then $\mathcal{M}(m, \pi_i(m)) = \mathcal{M}(m, \pi_{i-1}(m)) = 0$.

The mappings are inverses of each other. It is easy to check directly from the definition that if we start with a stable marriage P , convert it to a feasible pair $(\mathcal{M}, \mathcal{W})$, and convert it back into a stable marriage P' , then $P = P'$.

For the other direction, Lemma 40 shows that if $(\mathcal{M}, \mathcal{W})$ is a 0/1-valued feasible pair then for each man m , $\mathcal{M}(m, \cdot)$ consists of a positive number of 1s followed by 0s, and dually for each woman w , $\mathcal{W}(w, \cdot)$ consists of a positive number of 0s followed by 1s. Thus, given the fact that our rule of converting $(\mathcal{M}, \mathcal{W})$ to a stable marriage P indeed results in a marriage, it is clear that converting P back to a feasible pair results in $(\mathcal{M}, \mathcal{W})$. ◀

► **Lemma 43.** *Subramanian's algorithm returns the man-optimal and woman-optimal stable marriages. Furthermore, the matrices \mathcal{M}, \mathcal{W} at the end of Subramanian's algorithm coincide with the matrices \mathcal{M}, \mathcal{W} at the end of Algorithm 5.*

Proof. The monotonicity of \wedge and \vee shows that if we replace every $*$ in \mathcal{M}, \mathcal{W} with 0, then the resulting $(\mathcal{M}, \mathcal{W})$ is still a feasible pair; the same holds if we replace every $*$ with 1.

Lemma 41 and Lemma 42 together imply that the first output is the man-optimal stable matching, and the second output is the woman-optimal stable matching. Lemma 40 shows that at termination, the matrices \mathcal{M}, \mathcal{W} are matrix representations of intervals, hence they must coincide with the matrices at the end of Algorithm 5. ◀

► **Lemma 44.** *Subramanian's algorithm terminates after at most $2n^2$ iterations.*

Proof. Since there are $2n^2$ entries in both matrices, and at each iteration at least one entry changes from $*$ to 0 or 1, the algorithm terminates after at most $2n^2$ iterations. ◀

A formal correctness proof of Subramanian's algorithm can be found in [11].

7.3.5 MoSM and WoSM are AC^0 many-one reducible to CCV

In the remaining section, we will show that Subramanian's algorithm can be implemented as a three-valued comparator circuit.

First, since for each man m , the pair of values $\mathcal{M}_t(m, \pi_{i-1}(m))$ and $\mathcal{W}_t(\pi_{i-1}(m), m)$ is only used once to compute the two outputs $\mathcal{M}_t(m, \pi_{i-1}(m)) \wedge \mathcal{W}_t(\pi_{i-1}(m), m)$ and $\mathcal{M}_t(m, \pi_{i-1}(m)) \vee \mathcal{W}_t(\pi_{i-1}(m), m)$, and then each output is used at most once when updating $\mathcal{M}_{t+1}(m, \pi_i(m))$ and $\mathcal{W}_{t+1}(m, \pi_i(m))$. Thus the whole update rule can be easily implemented using comparator gates.

Second, we know that the algorithm converges within $2n^2$ iterations to a fixed point. Therefore, if we run the loop for exactly $2n^2$ iterations, the result would be the same. Hence, we can build a comparator circuit to simulate exactly $2n^2$ iterations of Subramanian's algorithm.

Finally, we can extract the man-optimal stable matching using a simple comparator circuit with negation gates. Recall that the logical values 0, $*$, 1 are represented in reality by pairs of wires with values (0, 0), (0, 1), (1, 1).

In the man-optimal stable matching, a man m is matched to $\pi_i(m)$ if $\mathcal{M}(m, \pi_i(m)) = 1$ and either $i = n$ or $\mathcal{M}(m, \pi_{i+1}(m)) \in \{0, *\}$. In the latter case, if the corresponding wires are (α, β) and (γ, δ) , then the required information can be extracted as $\alpha \wedge \beta \wedge \neg \gamma$.

► **Theorem 45.** *MoSM and WoSM are AC^0 many-one reducible to $CCV \neg$.*

Proof. We will show only the reduction from MoSM to $CCV \neg$ since the reduction from WoSM to $CCV \neg$ works similarly.

Following the above construction, we can define an AC^0 function that takes as input an instance of MoSM with preference lists for all the men and women, and produces a three-valued comparator circuit that implements Subramanian's algorithm, and then extracts the man-optimal stable matching. ◀

Corollary 28 and Theorems 31, 30 and 45 give us the following corollary.

► **Corollary 46.** *The ten problems MoSM, WoSM, SM, CCV, $CCV \neg$, THREE-VALUED CCV, 3LFMM, LFMM, 3VLFMM and VLFMM are all equivalent under AC^0 many-one reductions, where the equivalence of SM is with respect to the search problem version of the reduction defined in Section 2.3.*

Proof. Corollary 28 and Theorem 31 show that CCV, $CCV \neg$, THREE-VALUED CCV, 3LFMM, LFMM, 3VLFMM and VLFMM are all equivalent under AC^0 many-one reductions.

Theorem 45 shows that MoSM and WoSM are AC^0 many-one reducible to THREE-VALUED CCV. Theorem 30 also shows that 3LFMM is AC^0 many-one reducible to MoSM, WoSM, and SM. Hence, MoSM, WoSM, and SM are equivalent to the above problems under AC^0 many-one reductions. ◀

A Simplified proof of $NL \subseteq CC$

Each instance of the REACHABILITY problem consists of a directed acyclic graph $G = (V, E)$, where $V = \{u_0, \dots, u_{n-1}\}$, and we want to decide if there is a path from u_0 to u_{n-1} . It is well-known that REACHABILITY is NL-complete. Since a directed graph can be converted into a layered graph with an equivalent reachability problem, it suffices to give a comparator circuit construction that solves instances of REACHABILITY satisfying the following assumption:

The graph G only has directed edges of the form (u_i, u_j) , where $i < j$. (A.1)

The following construction from [11] for showing that $NL \subseteq CC$ seems more intuitive than the one in [16, 12]. Moreover, it reduces REACHABILITY to CCV directly without going through some intermediate complete problem, and this was stated as an open problem in [16, Chapter 7.8.1].

We will demonstrate the construction through a simple example, where we have the directed graph in Fig. 13 satisfying the assumption (A.1). We will build a comparator circuit as in Fig. 14, where the wires v_0, \dots, v_4 represent the vertices u_0, \dots, u_4 of the preceding graph and the wires ι_0, \dots, ι_4 are used to feed 1-bits into the wire v_0 , and from there to the other wires v_i reachable from v_0 . We let every wire ι_i take input 1 and every wire v_i take input 0.

We next show how to construct the gadget contained in each box. For a graph with n vertices ($n = 5$ in our example), the k^{th} gadget is constructed as follows:

- 1: Introduce a comparator gate from wire ι_k to wire v_0
- 2: **for** $i = 0, \dots, n - 1$ **do**
- 3: **for** $j = i + 1, \dots, n - 1$ **do**
- 4: Introduce a comparator gate from v_i to v_j if $(u_i, u_j) \in E$, or a dummy gate on v_i otherwise.
- 5: **end for**
- 6: **end for**

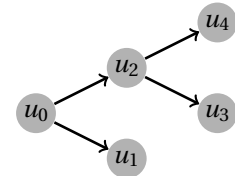
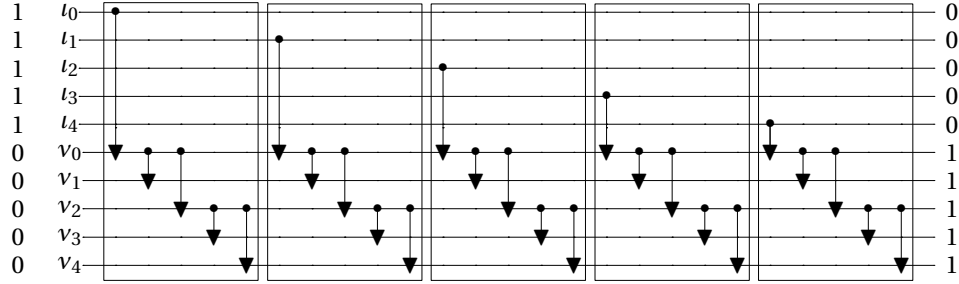


Figure 13

Note that the gadgets are identical except for the first comparator gate.

We only use the loop structure to clarify the order the gates are added. The construction can easily be done in AC^0 since the position of each gate can be calculated exactly, and thus all gates can be added independently from one another. Note that for a graph with n vertices, we have at most n vertices reachable from a single vertex, and thus we need n gadgets as described above. In our example, there are at most 5 wires reachable from wire v_0 , and thus we utilize the gadget 5 times.



■ **Figure 14** A comparator circuit that solves REACHABILITY. (The dummy gates are omitted.)

Intuitively, the construction works since each gadget from a box looks for the *lexicographically first maximal path* starting from v_0 (with respect to the *natural lexicographical ordering* induced by the vertex ordering v_0, \dots, v_n), and then the vertex at the end of the path will be marked (i.e. its wire will now carry 1) and thus excluded from the search of the gadgets that follow. For example, the gadget from the left-most dashed box in Fig. 14 will move a value 1 from wire l_0 to wire v_0 and from wire v_0 to wire v_1 . This essentially “marks” the wire v_1 since we cannot move this value 1 away from v_1 , and thus v_1 can no longer receive any new incoming 1. Hence, the gadget from the second box in Fig. 14 will repeat the process of finding the lex-first maximal path from v_0 to the remaining (unmarked) vertices. These searches end when all vertices reachable from v_0 are marked. Note that this has the same effect as applying the *depth-first search* algorithm to find all the vertices reachable from v_0 . Thus, the following theorem follows from the above construction.

► **Theorem 47** (Feder [12]). $NL \subseteq CC$.

References

- 1 K. Aehlig, S. Cook, and P. Nguyen. Relativizing Small Complexity Classes and Their Theories. In Jacques Duparc and Thomas Henzinger, editors, *Computer Science Logic*, volume 4646 of *Lecture Notes in Computer Science*, pages 374–388. Springer Berlin / Heidelberg, 2007.
- 2 E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO, Theoretical Informatics and Applications*, 30(1):1–21, 1996.
- 3 K.E. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference* 32, pages 307–314. ACM, 1968.
- 4 S. Cook and P. Nguyen. *Logical foundations of proof complexity*. Cambridge University Press, 2010.
- 5 S. A. Cook, D. T. M. Lê, and Y. Ye. Complexity Classes and Theories for the Comparator Circuit Value Problem. *arXiv*, abs/1106.4142, 2011.
- 6 T. Feder. A new fixed point approach for stable networks and stable marriages. *Journal of Computer and System Sciences*, 45(2):233–284, 1992.
- 7 T. Feder. *Stable Networks and Product Graphs*. American Mathematical Society, Boston, MA, USA, 1995.
- 8 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.
- 9 R. Greenlaw and S. Kantabutra. On the parallel complexity of hierarchical clustering and CC-complete problems. *Complexity*, 14(2):18–28, 2008.
- 10 R.M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

- 11 D. T. M. Lê, S. A. Cook, and Y. Ye. A Formal Theory for the Complexity Class Associated with the Stable Marriage Problem. In Marc Bezem, editor, *Computer Science Logic (CSL'11) - 25th International Workshop/20th Annual Conference of the EACSL*, volume 12 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 381–395, Dagstuhl, Germany, 2011.
- 12 E.W. Mayr and A. Subramanian. The complexity of circuit value and network stability. *Journal of Computer and System Sciences*, 44(2):302–323, 1992.
- 13 C. Moore and J. Machta. Internal Diffusion-Limited Aggregation: Parallel Algorithms and Complexity. *Journal of Statistical Physics*, 99:661–690, 2000.
- 14 K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.
- 15 V. Ramachandran and L.-C. Wang. Parallel algorithm and complexity results for telephone link simulation. In *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, pages 378–385, 1991.
- 16 A. Subramanian. *The computational complexity of the circuit value and network stability problems*. PhD thesis, Dept. of Computer Science, Stanford University, 1990.
- 17 A. Subramanian. A new approach to stable matching problems. *SIAM Journal on Computing*, 23(4):671–700, 1994.

Acknowledgment

We would like to thank Yuli Ye for his contribution in the early parts of this research. A portion of this work was done when the second and third authors received funding from the [European Community's] Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 238381. This work was also supported by the Natural Sciences and Engineering Research Council of Canada.